UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU EN ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

SDK per a la Representació d'Objectes en 3D

Autor: Ridouane Zaim Saikouk

Directors: Marc Bolaños i Petia Radeva
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 17 de Enero de 2024

UNIVERSITAT DE BARCELONA

Resum

Facultat de Matemàtiques i Informàtica

Departament de Matemàtiques i Informàtica

SDK per a la Representació d'Objectes en 3D

by Ridouane Zaim Saikouk

Creation of an SDK where the main goals are camera and frames information gathering and classification, the main frame of the capture detection and video creation applied to the food domain made by the frames captured using algorithms implemented in Android, is the main topic of this work. We have studied how to capture the correct and necessary images to build a representation of food based on the center of the scene and the angle created between the first frame and the center of the scene. To achieve this goal we have implemented several algorithms to detect, compute and update the center of the scene dynamically and also compute the angles and classify them into ranges in real time to achieve an optimal 3D representation. Furthermore, the Kalman filter has been implemented to provide more robustness to the data acquisition, preventing internal errors in obtaining information from the captured frame camera. Also, since the capture of frames is done dynamically, it allows us to improve motion errors made unconsciously by the user.

La creació d'un SDK on els principals objectius són la recopilació i classificació d'informació extreta de càmera i de fotogrames, la detecció de la imatge principal de la captura i la creació de vídeos formats per les imatges capturades mitjançant una sèrie d'algoritmes implementats en Android, aplicats al domini alimentari, són els temes principals d'aquest treball. Hem estudiat com capturar les imatges correctes i necessàries per construir una representació del menjar basada en el centre de l'escena i l'angle creat entre el primer fotograma i el centre de l'escena. Per assolir aquest objectiu, hem implementat diversos algorismes per detectar, calcular i actualitzar dinàmicament el centre de l'escena i també calcular els angles i classificar-los en rangs en temps real per obtenir una representació òptima en 3D. A més, s'ha implementat el filtre de Kalman per donar més robustesa a l'adquisició de les dades per evitar errors interns en l'obtenció de la informació de la càmera del fotograma capturat i també, com que la captura dels fotogrames es fa dinàmicament ens permet millorar errors de moviment fets inconscientment per l'usuari.

La creación de un SDK donde los principales objetivos son la recopilación y clasificación de información extraída de cámara y de fotogramas, la detección de la imagen principal de la captura y la creación de videos formados por las imágenes capturadas mediante una serie de algoritmos implementados en Android, aplicados al dominio alimentario, son los temas principales de este trabajo. Hemos estudiado como capturar las imágenes correctas y necesarias para construir una representación de la comida basada en el centro de la escena y el ángulo creado entre el primer fotograma y el centro de la escena. Para lograr este objetivo, hemos implementado varios algoritmos para detectar, calcular y actualizar dinámicamente el centro de la escena y también calcular los ángulos y clasificarlos en rangos en tiempo real para obtener una representación óptima en 3D. Además, se ha implementado el filtro de Kalman para dar una mayor robustez a la adquisición de datos, evitando errores internos en la obtención de información de la cámara del fotograma capturado. Además, dado que la captura de los fotogramas se realiza de manera dinámica, nos permite corregir errores de movimiento realizados inconscientemente por el usuario.

Agraïments

En primer lloc, m'agradaria agrair als meus tutors Petia Radeva i Marc Bolaños que m'han guiat de manera continuada durant el desenvolupament del projecte. A més, m'agradaria fer menció especial a Leonel Viera que també m'ha guiat de manera excepcional. Han sigut de gran ajuda per donar-me consells de com afrontar els problemes i testejar-los, per consells més tècnics en la implementació dels algoritmes cal destacar l'ajuda de Maria Fernanda Herrera.

Finalment, vull expressar el meu agraïment en general a l'equip de LogMeal pel suport constant rebut i l'experiència guanyada.

Glossari

UB: Universitat de Barcelona

Spin-off: Projecte nascut com a extensió d'un altre d'anterior.

API: Una interfície de programació d'aplicacions (en anglès *Application Programming Interface*, API) és un conjunt d'indicacions, quant a funcions i procediments, ofert per una biblioteca informàtica o programoteca per ser utilitzat per un altre programa per interaccionar amb el programa en qüestió.

Interfície d'usuari: La interfície d'usuari és el medi amb què l'usuari pot comunicar-se amb una màquina, un equip o una computadora, i comprèn tots els punts de contacte entre l'usuari i l'equip; normalment solen ser fàcils d'entendre i fàcils d'accionar.

Branca: En Git, una branca (o *branch* en anglès) és una línia de desenvolupament independent que permet als desenvolupadors treballar en funcionalitats o correccions de bugs sense afectar directament la branca principal del codi (sovint anomenada *master* o *main*). Les branques en Git proporcionen un entorn aïllat per a les modificacions, i els canvis realitzats en una branca no afecten les altres fins que s'integrin.

Bug: En el context de la programació informàtica, un *bug* fa referència a un error o defecte en el codi d'un programa que provoca un comportament inesperat o incorrecte.

Endpoint: Es refereix a un punt d'entrada o interfície d'un servei. Aquest terme és freqüent en l'arquitectura REST (*Representational State Transfer*), que és un estil d'arquitectura àmpliament utilitzat per al desenvolupament de serveis web.

SDK: Un SDK (*Software Development Kit*) és un conjunt d'eines de programari que proporciona un entorn de desenvolupament per a la creació d'aplicacions per a una plataforma o tecnologia específica.

Deep learning: El *Deep Learning* (Aprenentatge Profund) és una subàrea de l'aprenentatge automàtic (machine learning) que es basa en xarxes neuronals artificials profundes per modelar i resoldre tasques complexes.

Intel·ligència artificial o IA: La Intel·ligència artificial (IA) és un camp de la informàtica que se centra en la creació de sistemes capaços d'executar tasques que requereixen intel·ligència humana. Aquests sistemes es dissenyen per simular processos cognitius humans, com ara aprenentatge, raonament, resolució de problemes, comprensió del llenguatge natural i percepció visual.

RA: realitat augmentada

ArCore: ARCore és una plataforma de realitat augmentada (RA) desenvolupada per Google per a dispositius Android. Aquesta tecnologia permet als desenvolupadors crear aplicacions que combinen elements virtuals amb el món real utilitzant la càmera del dispositiu i la seva percepció de l'entorn.

Endpoint food quantity estimation: *endpoint* per estimar la quantitat de menjar.

Computer vision: La *Computer Vision* (Visió per Computador) és una àrea de la informàtica que se centra en el desenvolupament de sistemes que permeten als

ordinadors interpretar, analitzar i comprendre imatges i vídeos de la mateixa manera que ho fan els éssers humans. Aquesta disciplina abasta diverses tasques, des de la detecció d'objectes fins a la reconeixement facial i la segmentació d'imatges.

Transposada d'una matriu: La transposada d'una matriu és una operació que implica intercanviar les files i les columnes de la matriu original.

Framework: Un *framework* és una estructura o plataforma conceptual que proporciona eines, llibreries i altres components per facilitar el desenvolupament i la implementació d'aplicacions o projectes. Aquestes estructures ofereixen un conjunt de regles, pràctiques i convencions per ajudar els desenvolupadors a crear de manera eficient i estandarditzada.

APK: Un APK (*Android Package*) és el format de fitxer utilitzat per al sistema operatiu Android per distribuir i instal·lar aplicacions. És essencialment un fitxer comprimit que conté tots els elements necessaris per a una aplicació Android, com ara el codi font, recursos, imatges, fitxers de configuració i altres elements requerits per a l'aplicació.

Bearer Token: Un *Bearer Token* és un tipus de token d'autorització que s'utilitza en el context de l'autenticació i autorització en sistemes informàtics, especialment en entorns web i API

Hint: En el context d'Android, un *hint* (pista o indicació) en un element de text, com ara un *TextInputEditText*, es refereix a un text temporal o indicació visual que es mostra a l'usuari dins del camp de text quan aquest està buit.

Postman: Postman és una eina de desenvolupament que permet als desenvolupadors realitzar peticions HTTP, provar API i col·laborar en el desenvolupament d'aplicacions. Es tracta d'una plataforma que ofereix diverses funcionalitats per simplificar i accelerar el procés de desenvolupament d'API.

Terminal: Terminal es refereix a una interfície d'usuari textual que permet als usuaris interactuar amb un sistema informàtic mitjançant comandes de text. Les terminals són àrees on els usuaris poden escriure i executar comandes per realitzar tasques diverses.

Main RGB image o imatge principal: En el context d'una captura, és la imatge més propera a l'angle de 90 graus.

Focal length: La *focal length* (distància focal) és un paràmetre important en òptica i fotografia que es mesura en mil·límetres (mm). Es refereix a la distància des de la lent de la càmera fins al punt on els raigs de llum es converteixen i es formen en una imatge nítida al sensor o pel·lícula.

Principal point: El *punt principal* (principal point en anglès) en fotografia i visió per computador és un punt especial en la imatge capturada que es troba al centre de la imatge resultant quan s'utilitza una càmera ideal. Aquest punt és la intersecció de l'eix òptic amb el pla de la imatge.

Camera pose: La *camera pose* (posició de la càmera) es refereix a la posició tridimensional i orientació d'una càmera respecte a un sistema de coordenades global. Aquesta informació descriu la ubicació i l'orientació de la càmera en l'espai tridimensional i és fonamental en la visió per computador i la realitat augmentada.

Diagrama de clases: Un diagrama de classes és una representació visual d'una estructura de classes i les relacions entre elles en un sistema orientat a objectes.

Diagrama de flux: Un diagrama de flux és una representació visual d'un procés o sistema que mostra la seqüència d'activitats i com les dades es mouen a través d'aquestes activitats. Aquests diagrames són utilitzats per descriure de manera detallada els passos d'un procés i com interactuen els diferents elements involucrats.

Anchor: En el context d'ARCore, el terme *anchor* es refereix a un concepte clau usat per ancorar objectes o informació virtuals en el món real. Les "anchors"serveixen com a punts de referència estables en l'entorn físic, permetent que els objectes de realitat augmentada estiguin vinculats a ubicacions específiques.

Índex

Resum iii

Agraïments v

Glossari vii

1 Introducció 1
- 1.1 Context . . . 1
- 1.2 Motivació . . . 4
  - 1.2.1 Àmbit general . . . 4
  - 1.2.2 Àmbit personal . . . 4
- 1.3 Organització . . . 5
- 1.4 Objectius . . . 5
- 1.5 Planificació . . . 9

2 Anàlisi i disseny 11
- 2.1 Diagrama de flux . . . 11
- 2.2 Anàlisis de requisits . . . 14
- 2.3 Diagrama de classe . . . 17
- 2.4 Interfície d'usuari . . . 20
  - 2.4.1 Interfície d'usuari conceptual . . . 22

3 Implementació 25
- 3.1 Obtenició de les dades del frame . . . 25
- 3.2 Posició de la càmera . . . 26
- 3.3 Filtre de Kalman . . . 27
  - 3.3.1 Context . . . 27
  - 3.3.2 Paràmetres . . . 29
  - 3.3.3 Inicialització . . . 29
  - 3.3.4 Predicció . . . 30
  - 3.3.5 Correcció . . . 30
- 3.4 Centre de l'escena . . . 31
  - 3.4.1 Context . . . 31
  - 3.4.2 Inicialització . . . 32
  - 3.4.3 Actualització del centre de l'escena . . . 32
  - 3.4.4 Algoritme per calcular el punt més proper entre dues línies: 34
  - 3.4.5 Actualització del centre de l'escena . . . 36
- 3.5 Computació de l'angle . . . 36
  - 3.5.1 Primera versió . . . 36
  - 3.5.2 Segona versió . . . 37
- 3.6 Generació dels fitxers . . . 38
  - 3.6.1 Seqüència de Vídeo . . . 38

3.6.2 Arxiu JSON 39
3.6.3 Imatge principal 40
3.7 Procediment complet 40
3.8 Crida a l'API 41
3.9 Consideracions 42

4 Resultats **43**

5 Conclusions **49**

6 Treball futur **51**

Bibliografia **53**

Capítol 1

Introducció

El treball de fi de grau *Extracció de representació d'imatges en 3D* ha estat realitzat en col·laboració amb una empresa anomenada LogMeal, i ha estat dirigit per Petia Radeva, tutora de la UB, i Marc Bolaños, ambdós cofundadors d'aquesta.

1.1 Context

LogMeal és una aplicació d'intel·ligència artificial per reconèixer menjar i els valors nutricionals d'aquest a través d'una captura realitzada per un usuari de l'aplicació. D'aquesta manera LogMeal busca simplificar i automatitzar el procés d'aconseguir calcular la quantitat de menjar amb precisió i oferir una informació nutricional dels aliments detallada. Cal destacar que LogMeal, dins de l'àmbit alimentari, utilitza tecnologies com *deep learning* i algoritmes de *computer vision artificial intelligence*. Més informació a Bhalaji Nagarajan, 2021, Nutritional Monitoring in Older People Prevention Services.

On i quan va néixer LogMeal? LogMeal és una organització nascuda l'any 2012 com a extensió de la Universitat de Barcelona, és un *spin-off* de la UB.

"Analitza en temps real les teves menjades simplement fent una foto. Detectem cada aliment al plat o safata, avalua la quantitat servida i estima tots els valors nutricionals clau, tant micro com macro. L'API més avançada en IA alimentària, seguiment d'aliments i generació de dietaris. La millor solució d'API de detecció d'imatges d'aliments per al teu negoci, proporcionant informació d'imatges d'aliments, incloent-hi tipus d'aliment, grups d'aliments, plats, ingredients o receptes, així com informació nutricional (32 nutrients, micro i macro). La millor solució per fer un seguiment exhaustiu dels teus clients, esportistes o pacients. API completa en 35 llenguatges de programació". Més informació a, LogMeal, 2017, LogMeal Food AI.

Els serveis que ofereix LogMeal són LogMeal API, LogMeal APP, LogMeal PLATFORM i LogMeal KIOSK.

- LogMeal API és l'API per al reconeixement d'aliments i la detecció d'aliments basada en imatges. Inclou etiquetatge semàntic, que inclou la identificació del grup d'aliments, del plat i dels ingredients, així com l'anàlisi de la informació nutricional. L'API permet que les funcionalitats i algoritmes d'intel·ligència artificial, visió per ordinador i algoritmes d'aprenentatge profund implementats per LogMeal estiguin disponibles.

- LogMeal APP és l'aplicació de LogMeal per a dispositius Android i iOS, et permet ser més conscient de les teves ingestes, t'ajuda a aconseguir una dieta conscient i saludable, i a millorar la teva qualitat de vida. LogMeal APP utilitza LogMeal API i el seu funcionament és el següent: Només amb fer una foto del teu àpat, LogMeal el reconeixerà en temps real i et proporcionarà els següents detalls:
  - Llista d'ingredients (Presenta els ingredients del plat de la imatge així com les seves quantitats estàndard. Malgrat això, sempre es pot modular les quantitats i LogMeal recalcularà automàticament tots els valors).
  - Informació detallada sobre la nutrició (32 indicadors nutricionals com energia, macronutrients, fibra, colesterol, vitamines i minerals).

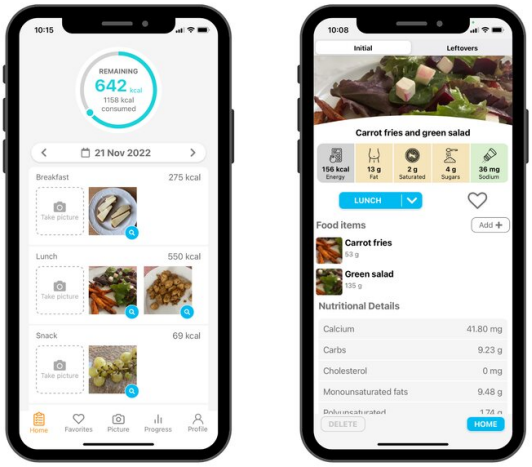

FIGURA 1.1: Exemple de la interfície d'usuari de l'APP. Font: LogMeal, 2017, LogMeal Food AI

- LogMeal PLATFORM és una eina web dissenyada per als experts com nutricionistes i sanitaris que permet entre altres funcionalitats: l'establiment d'objectius nutricionals, gestió, seguiment, monitoratge i visualització de dades (alertes i indicadors), permet visualitzar gràfics detallats d'indicadors de nutrients, visualització de rangs de dades, visualitzar un resum de la ingesta d'un usuari i de les restes de la ingesta.
- LogMeal KIOSK integra el programari de LogMeal en el quiosc d'un soci fabricant, on col·laboren utilitzant el seu maquinari per integrar fàcilment aquesta solució en qualsevol restaurant o client. Com funciona? El client simplement agafarà per posar un exemple la safata amb els aliments que vulgui i la posarà al quiosc. Aquest prendrà una foto i els algoritmes al núvol reconeixen cada aliment en temps real. El sistema és molt adaptable a qualsevol restaurant i pot aprendre en temps real els aliments que se serveixen. Pot detectar els aliments, així com el preu, les combinacions de preus del menú i ofereix diferents solucions de pagament per al client.

FIGURA 1.2: Exemple del maquinari en un restaurant self-service. Font: LogMeal, 2017, LogMeal Food AI

Situant-nos en aquest context en què LogMeal busca ser el pioner en aquest àmbit, treballarem principalment en la implementació integral d'un SDK en *Android* i de manera opcional es proposa incorporar i integrar aquest SDK d'*Android* en un altre llenguatge de programació anomenat *React-native*. Per tant, treballarem per implementar un equip de desenvolupament de programari per dotar a LogMeal API d'un conjunt d'eines útils per al funcionament d'un dels algoritmes més importants que utilitza que és l'anomenat *food quantity estimation*[1]

Aquest projecte aportarà un valor afegit interessant no només a LogMeal sinó a qualsevol persona o empresa que vulgui contractar aquest servei per usar el SDK i observem que el plantejament d'aquest treball de fi de grau dins d'aquest context és ambiciós i representa una part important per a LogMeal. Cal destacar que el SDK implementat s'integrarà a l'API d'aquest. Més endavant s'explicarà amb el màxim detall els objectius principals d'aquest projecte.

FIGURA 1.3: Logotip de LogMeal. Font: LogMeal, 2017, LogMeal Food AI

[1] És l'algoritme que permet calcular la quantitat o volum de menjar a partir de fotogrames.

1.2 Motivació

1.2.1 Àmbit general

La motivació d'aquest treball radica en contribuir a millorar la societat en la qual vivim aportant unes eines clau per ajudar a estimar el menjar. Convé subratllar la importància d'estimar la quantitat de menjar quan volem analitzar la dieta d'una persona, ja que, l'OMS afirma que 6 de cada 10 malalties que patim està directament relacionada amb l'alimentació. Més informació a La Vanguardia, 2018, Percentatge de malalties relacionades amb l'alimentació.

Una possible millora per reduir aquest percentatge és establir una dieta flexible. Moltes vegades el metge o nutricionista ens dona un varem per les quantitats o consells com per exemple, no passar-se de 25 grams de sucre al dia, que moltes vegades no se segueixen per falta d'eines per obtenir informació nutricional. Cal destacar també que la gent tingui nutricionista o no, sol tenir problemes per estimar les quantitats, ja sigui, perquè li fa mandra pesar els aliments cada dia o bé per altre motiu. Les malalties que es poden adquirir per una mala alimentació són les següents: diabetis, osteoporosis, càncer de còlon, sobrepès i obesitat, hipertensió arterial, malalties cardiovasculars, etc. Més informació a Savino, 2011, Obesitat i malalties no transmissibles relacionades amb la nutricio

Per tant, ens proposem implementar un conjunt d'eines per estimar el menjar amb l'objectiu d'informar a l'usuari sobre la informació nutricional i quantitat d'aliment que està ingerint. Aquesta informació pot ser usada per un professional especialitzat per analitzar si està seguint una bona dieta en l'àmbit alimentari o no, i així, corregir-la en cas necessari. És important destacar que la informació que es pot assolir gràcies a les eines d'aquest projecte i els algoritmes de LogMeal són dades reals del menjar que l'usuari ingereix.

1.2.2 Àmbit personal

Des del darrer any tenia clar que volia fer un treball de fi de grau relacionat amb el desenvolupament d'una aplicació de dispositiu intel·ligent, en específic implementar una aplicació en Android o iOS perquè em va agradar l'experiència d'una assignatura anomenada Projecte Integrat de Software. La intenció era relacionar el projecte amb les meves aficions, el gimnàs, el futbol i els esports de contacte. No obstant això, no tenia clara la temàtica exacta del projecte fins que em van proposar ajudar a LogMeal en el desenvolupament d'un SDK en Android.

L'any 2018 vaig començar la universitat i just també és en aquest any que vaig donar els meus primers passos en el gimnàs fins a la pandèmia però sense ser constant amb la dieta. Obtenir resultats no només és anar al gimnàs, hi ha més factors que entren en joc i després de fer un treball d'investigació vaig descobrir la importància de dormir i alimentar-se correctament. Tothom sap que per dormir i descansar bé es necessiten 8 hores aproximadament, però tenir consciència de com ens estem alimentant no ho té tothom tan clar, òbviament segons els objectius que tingui una persona l'exigència en l'àmbit alimentari és una o altra.

En aquest període d'investigació vaig descobrir MyFitnessPal una aplicació que d'entre altres funcionalitats interessants que té és majoritàriament coneguda per emmagatzemar i llistar els aliments que consumeixes diàriament, les seves calories, els macronutrients i els micronutrients. A nivell gimnàs és interessant perquè

et permet fer un resum de les calories que estàs ingerint, i a més et permet observar fàcilment els grams dels 3 principals macronutrients que són les proteïnes, els hidrats de carboni i les grasses. L'inconvenient principal d'aquesta aplicació és que has d'introduir manualment tots els aliments, cosa que fa que el procés d'obtenir aquest resum diari sigui enrevessat. Aquest inconvenient és justament el que intenta solucionar LogMeal, és tan simple com fer una captura de fotogrames de per exemple el berenar i automàticament et llista els aliments i els seus valors nutricionals, és ràpid, senzill i el més important, estalvies temps tant per mesurar els aliments com per introduir manualment els aliments en una aplicació. Més informació sobre MyFitnessPal a: Lee, 2005, Una bon estat de salut comença per una bona alimentació

Treballar en aquest projecte per una aplicació dotada de tecnologies que hem usat durant la carrera que dona solució a un dels problemes que he tingut en el passat, i aportar solucions i eines a una part important de LogMeal van ser els principals motius de la meva motivació.

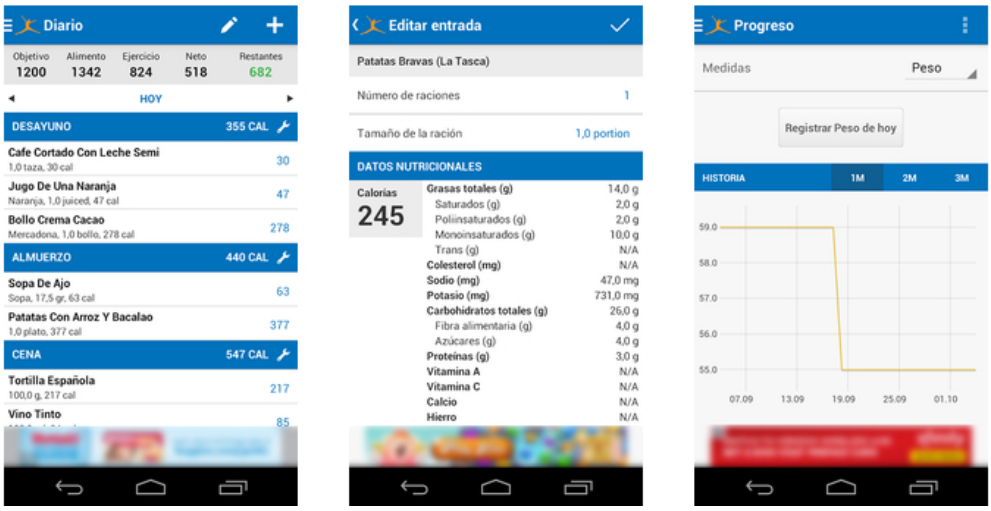

FIGURA 1.4: Imatge que mostra aliments introduïts manualment a MyFitnessPal. Font: Moya, 2014, MyFitnessPal

1.3 Organització

A continuació explicarem els objectius inicials i finals detalladament, i per concloure la introducció explicarem la planificació per la implementació dels nostres objectius. En el Capítol 2 parlarem de l'anàlisi, el disseny i la implementació del nostre SDK. Posteriorment, en el Capítol 3 explicarem les proves realitzades, en els capítols finals analitzarem els resultats obtinguts i avaluarem si hem validat i satisfet els objectius. Finalment, conclourem aquest treball amb unes conclusions del projecte i proposarem el treball pendent pel futur.

1.4 Objectius

Com ja hem avançat anteriorment el nostre objectiu principal és dotar a l'API de LogMeal d'un SDK implementat en Android, específicament pels *endpoints* de

Food Quantity Estimation, que són els *endpoints* relacionats amb l'aplicació de segmentació d'aliments i estimació de la quantitat a les imatges. Per tant, l'objectiu més important és realitzar una captura de manera dinàmica de X fotogrames, on cada fotograma és capturat en un rang d'angles específic. A més, també es du a terme la captura de la imatge principal del procediment de manera dinàmica. Finalment, es du a terme la creació d'un vídeo a partir dels fotogrames específics capturats i la creació d'un fitxer JSON que conté informació important de la captura que en els següents capítols explicarem amb més detall. L'objectiu final pel qual hem d'obtenir aquestes imatges és perquè l'API pugui obtenir una representació tridimensional de l'escena del menjar, i així poder fer una estimació de la quantitat a partir d'un model 3D.

Tot aquest procediment s'utilitzarà per a l'usuari final de LogMeal per fer la captura del menjar, per aquest motiu és molt rellevant estudiar quina és la millor opció en termes d'usabilitat. Per tant, hem de ser curosos amb l'exigència en el marge d'error que volem proposar a l'usuari pel que fa a trajectòria de moviment i hem de ser flexibles amb el nombre de fotogrames capturats correctament, tot mantenint una relació coherent entre usabilitat i resultat final adient.

Cal destacar que l'*endpoint* de *food quantity estimation* de l'API a la qual dotarem del SDK s'executa en segon pla; això significa que immediatament després que finalitzi la sol·licitud de reconeixement d'aliments, encara no es calcularà la quantitat d'aliments. Si es fa servir qualsevol *endpoint* que recuperi ingredients, informació nutricional o qualsevol altra dada relacionada amb la ingesta, es presentaran quantitats d'aliments estàndard. Després d'alguns segons, immediatament després que acabi l'estimació de la quantitat, tots els valors per a totes les crides als punts finals s'actualitzaran automàticament.

A més, qualsevol *endpoint* i procediment intern que es basi en la quantitat d'aliments utilitza per defecte la quantitat d'aliments estimada automàticament (per exemple, definició d'ingredients o estimació d'informació nutricional). S'ha de tenir en compte que els resultats de segmentació podrien no ser directament aplicables a la mida de la imatge carregada. Les imatges podrien redimensionar-se a vegades segons les necessitats dels algorismes de reconeixement d'aliments usats.

Per calcular la quantitat d'aliments, l'*endpoint* necessita que li proporcionem una seqüència de vídeo. El vídeo s'ha d'extreure utilitzant el SDK que nosaltres implementarem. A més necessitarà que li enviem uns altres inputs com una imatge i un JSON amb tota la informació relacionada amb la captura dels fotogrames.

Convé subratllar que per usar l'*endpoint* de *food quantity estimation* en l'API de LogMeal és important recordar té diversos paràmetres en el cos de la sol·licitud ('image', 'sequence', 'cam_focal_length', 'cam_principal_point', 'main_rgb_idx', 'camera_pose' i 'version') que estan a 'multipart/form-data'.

A continuació llistarem els *inputs* per l'API:

- Seqüència de vídeo.
- Imatge.
- JSON.

Aquests arxius són el resultat d'un procediment llarg d'investigació i de la creació d'una lògica per sincronitzar tots els algoritmes implementats que explicarem

més endavant. Ens proposem d'exposar les tasques a realitzar per satisfer els nostres objectius:

- Treball d'investigació sobre la llibreria ArCore. Més informació a: Google, 2018, ArCore: funcionament, avantatges, inconvenients, etc.
- Implementació d'un mètode per obtenir la informació essencial de la càmera.
- Implementació d'un mètode per guardar, en memòria, un fotograma (partint de la càmera).
- Realització en Android d'un model que permeti fer captures de X fotogrames.
- Lògica per guardar els arxius d'una mateixa captura en una carpeta única.
- Implementació de la lògica per inicialitzar el centre d'una escena en una captura.
- Implementació d'un mètode per actualitzar el centre de l'escena dinàmicament.
- Implementació d'un mètode per aplicar un *weight* al centre de l'escena.
- Implementació d'un mètode per inicialitzar *Kalman filter*. Més informació a: Becker, 2023, Introducció a Kalman filter
- Implementació d'un algoritme per calcular l'angle a partir dels atributs *altitude* i *azimuth* (versió inicial de l'algoritme).
- Implementació d'un algoritme per calcular l'angle a partir del centre de l'escena i la primera càmera o fotograma (versió final de l'algoritme).
- Investigació i implementació sobre quin és el millor rang d'angles en termes d'usabilitat per a realitzar una captura de manera còmoda.
- Implementació de la seqüència de vídeo a partir d'una sèrie de fotogrames seleccionats.
- Crear una estructura de dades per emmagatzemar tota la informació rellevant per a la creació del JSON final.
- Implementació d'un mètode per seleccionar la *main image* que és una imatge que definim de la següent manera: La *main RGB image* és la imatge que s'obté quan l'angle de la captura del fotograma estigui més proper a l'angle de 90 graus, és la imatge més important de la captura.
- Implementació de la lògica de l'algoritme final on es posa en relació tots els algoritmes creats.

Pel que fa a l'objectiu opcional d'integrar aquest SDK en *React-native*, tot i que era un objectiu inicial opcional i extra, convé fer ressaltar que no hi ha hagut temps a realitzar-ho pels següents motius:

- En primer lloc, convé destacar que durant el desenvolupament en *Android* vam observar que de manera interna en aquest llenguatge de programació existien errors en com detectava les posicions de la càmera, per aquest motiu s'ha hagut d'investigar i implementar un filtre per corregir aquests errors. Cal destacar la importància de solucionar aquests errors, ja que, si

tenim errors interns d'*Android* i li afegim errors que pugui dur a terme l'usuari de manera inconscient en realitzar la trajectòria de moviment per fer la captura, el resultat final que enviem a l'API no obtindria un bon resultat per l'algoritme *food quantity estimation*. Solucionar-ho no ha sigut tasca senzilla, encara més, hi ha hagut un grau de complexitat en implementar Kalman *filter* en Java que ha fet alentir la implementació.

- L'altre motiu que ha alentit encara més la implementació ha sigut que l'algoritme que volíem utilitzar inicialment per calcular els angles (que era molt més complex que el que finalment vam implementar) no era del tot correcte, i entre buscar un altre algoritme i implementar-ho de nou, i canviar la lògica perquè funcioni en conjunt amb la resta d'algoritmes va fer que s'endarrerís la implementació.

Finalment, s'ha de tenir en compte que l'objectiu del *framework React-native* és dotar de les eines primordials per tal de crear codi que sigui compatible tant per *Android* com per *Swift (o iOS)*, i com que un company es va unir al projecte per fer el seu treball de fi de grau en iOS, vam considerar que era millor proposar una implementació òptima i robusta en *Android* i deixar de banda el desenvolupament en *React-native* que tal com s'ha dit perd la seva gràcia en el moment que hi ha un altre company realitzant el desenvolupament en iOS.

1.5 Planificació

La metodologia que hem emprat per satisfer aquests objectius és partir de tasques més petites i anar incorporant-les entre elles d'aquesta manera s'aconsegueix atacar el problema de manera més senzilla. A més, treballar d'aquesta manera ens permet l'opció de realitzar tasques independents entre elles.

Per acomplir les tasques he estat supervisat i en contacte diari sobretot al principi amb Leonel Viera, on em recomanava diferents propostes per encarar de manera més senzilla les primeres tasques. A més, em va ajudar a fer com una mena de *onboarding* per entendre la filosofia de LogMeal, com també per presentar-me els repositoris de GitLab on hem pujat tot el codi implementat (el codi final només, les altres implementacions que no hem fet servir han sigut descartades, tot i que parlarem d'aquestes més endavant) i per explicar-me un codi inicial del qual partíem.

El punt següent tracta de les reunions setmanals que hem tingut amb l'equip de LogMeal des que vam iniciar el projecte a finals de juny de 2023. Cada dimarts hem tingut una reunió d'una durada aproximada d'una hora amb Marc Bolaños (CTO i *Co-Founder*), Eric Verdaguer (CEO i *Co-Founder*) i Leonel Viera (director de Panalsoft) per comentar les tasques fetes, resoldre dubtes i proposar les següents tasques a realitzar. Més tard es va unir a les reunions un altre company que està duent a terme una versió del SDK per iOS.

Pel que fa a les qüestions més tècniques he estat en contacte amb Maria Fernanda, especialista en *Image Processing* i *Computer Vision*, que ha sigut un suport molt important per la implementació dels algoritmes.

D'altra banda, amb l'objectiu de fer un seguiment més exhaustiu de les tasques que s'han fet i per comentar les tasques que queden pendents s'han fet diverses reunions, d'una durada més llarga, amb Petia Radeva i Marc Bolaños.

Un altre punt important és comentar el programari usat:

- Llenguatges de programació usats: Java i Android.
- Comunicació per escrit o reunions informals: Slack.
- Reunions setmanals i reunions més formals: Google Meet.
- Gestor de repositoris: GitLab.

Finalment, mostrarem un diagrama de *Gantt* on es visualitzen totes les tasques i la seva durada en setmanes. Les barres indiquen la durada i al costat de la barra es troba el nom de la tasca específica. El color blau fosc indica que la tasca està finalitzada, el color blau fluixet indica que és una tasca que es va fer, però que no es va considerar pel projecte final perquè ha sigut millorada i el color vermell indica una tasca no finalitzada. Visualització del diagrama:

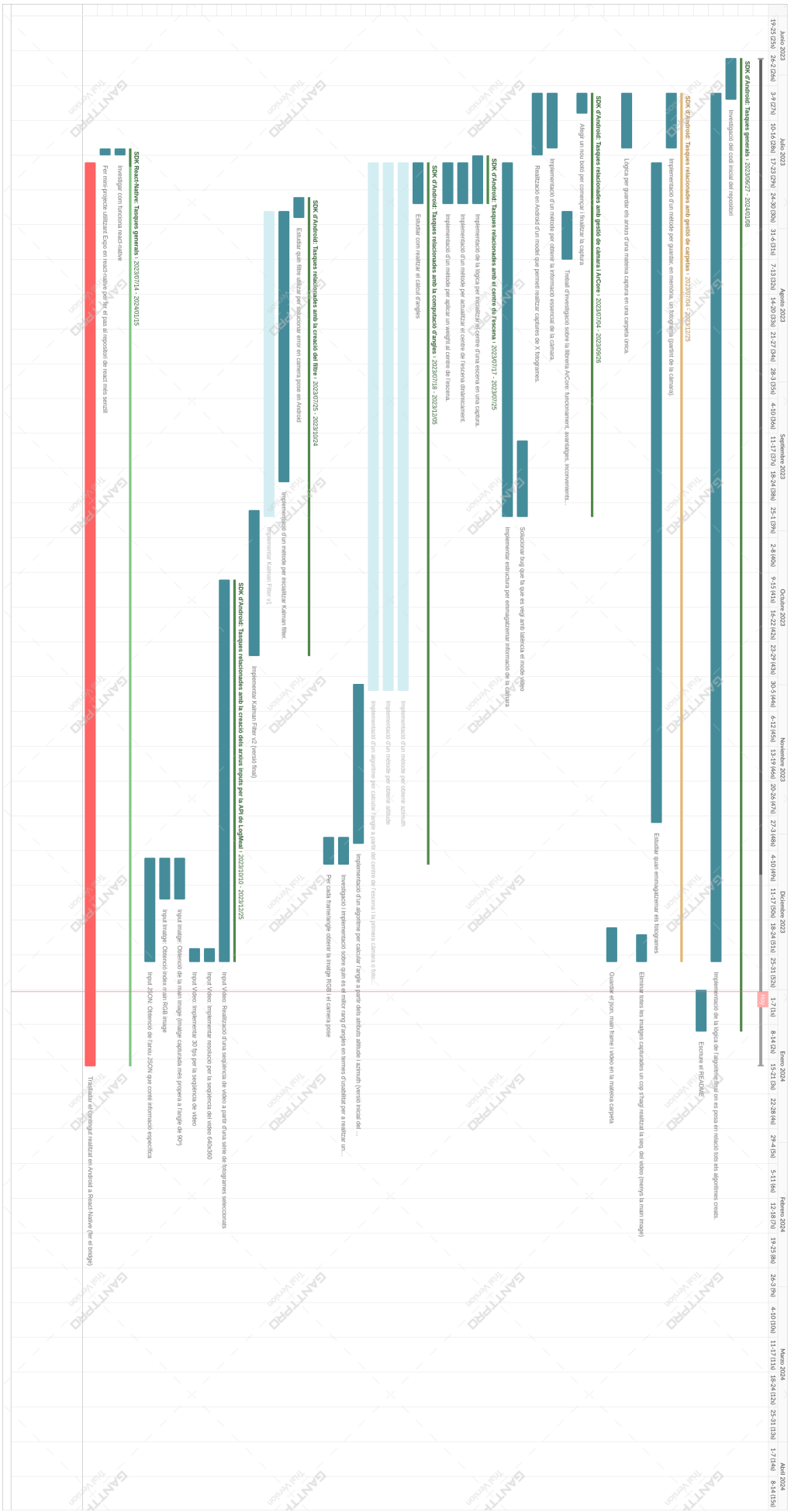

FIGURA 1.5: Diagrama de *Gantt*

Capítol 2

Anàlisi i disseny

En aquest capítol explicarem en detall l'anàlisi i el disseny del nostre programa. En la secció 2.1, per observar de manera genèrica com funciona la nostra aplicació començarem parlant del diagrama de flux que és una representació gràfica d'un algoritme o procés. En aquest diagrama explicarem quins són els passos que se segueixen en el codi des que l'usuari pren la decisió de començar la captura fins que les dades s'envien a l'API, tot explorant les sortides esperades per cada casuística de l'execució. Per fer els diagrames s'ha utilitzat: Ltd, 2000, Draw Io

En la següent secció 2.2 explicarem quins requisits identifiquem i considerem que són importants per "educar"a l'usuari sobre com hauria d'interactuar amb l'aplicació per obtenir una bona representació. En altres paraules, farem una anàlisi de requisits.

Posteriorment, en la secció 2.3 mostrarem un diagrama de classes del codi implementat, on breument donarem unes pinzellades sobre per què s'ha estructurat el codi del projecte d'aquesta manera.

En la darrera secció 2.4 parlarem de com hauria de ser la interfície d'usuari. Convé destacar que és un dels punts pendents per treballar en el futur, ja que, malgrat que no s'hagi pogut implementar la interfície d'usuari, hi ha hagut un treball important d'investigació sobre com hauria de ser aquesta interfície perquè l'usuari en tot moment estigués informat de l'estat real i actual de la captura.

2.1 Diagrama de flux

En el següent diagrama de flux es mostren els diferents passos que succeeixen des que l'usuari prem el botó per començar una nova captura. En primer lloc, s'inicialitzen les variables necessàries per a funcionament del procés. Concretament, s'estableixen els rangs dels angles, és a dir, cada quants angles s'ha de realitzar una captura d'un fotograma, a més, s'inicialitza el centre amb uns valors generals predefinits i finalment s'inicialitza la trajectòria que contindrà tota la informació sobre els *camera pose* detectats.

A continuació es comprova si el nombre de fotogrames capturats és igual o major al nombre de fotogrames desitjats, en el cas afirmatiu és el cas ideal que s'espera, ja que, significaria que s'ha fet el 100% de la captura i que a més, s'ha obtingut tota la informació necessària per a l'API que posteriorment és processada per obtenir una òptima representació 3D, tot seguit l'execució s'atura. En cas negatiu, s'aconsegueix el *camera pose* del fotograma actual per continuar amb l'execució.

Tal com s'ha dit un cop s'obté el *camera pose* actual, es pregunta si aquest pot ser corregit o no perquè pot ser l'usuari ha realitzat un moviment brusc o per algun motiu s'ha desviat massa de la trajectòria. Per una banda, si aquest *camera pose* pot ser corregit, vol dir que es pot aplicar el filtre de Kalman per millorar-lo o bé que ja és un valor correcte que no necessita ser millorat, per l'altre, l'execució s'atura i s'analitzen diferents casuístiques. Si ja s'ha capturat el *main RGB frame* i s'han capturat en total més de 30 fotogrames, vol dir que tot i que no s'hagi fet el 100% de la captura, hi ha suficient informació per enviar-ho a l'API de LogMeal. Més endavant en la secció 2.2 explicarem més detalls. Si no se satisfan aquests mínims exigits, es descarta tota la captura feta perquè es considera que és informació insuficient per poder enviar a l'API.

En el cas que el *camera pose* sigui correcte o pot ser corregit amb el filtre de Kalman, es realitza l'actualització del centre de l'escena fent servir el centre de l'escena anterior (en el cas de la primera iteració del procés, tal com s'ha dit anteriorment, el centre té un valor per defecte) i el *camera pose* actual. Posteriorment, s'executa un mètode per dotar al centre d'un pes. Aquest centre és el que s'utilitza per calcular posteriorment l'angle. L'angle es calcula usant el centre de l'escena i el primer *camera pose*.

Un altre punt important és comprovar si l'angle calculat està comprès en el rang d'angles, per exemple, si l'angle és 90 i el rang d'angle actual és [89.5, 91.5] s'emmagatzemaria tota la informació necessària d'aquest fotograma en la nostra estructura de dades i començaríem la iteració del programa de nou, és a dir, tornaríem a comprovar si el nombre de fotogrames ja capturats és igual o major al nombre de fotogrames desitjat. Altrament, per exemple, si l'angle fos 93 i el rang d'angles el mateix que l'exemple anterior aleshores es descartaria les dades del fotograma actual i començaríem la iteració a partir de l'obtenció d'un altre *camera pose*. De manera paral·lela, independentment de si l'angle està comprès o no en el rang d'angles actual, s'ha d'actualitzar la trajectòria.

Finalment, és important recordar que l'usuari en qualsevol moment de l'execució té l'opció de prendre un botó per aturar aquesta. Si l'usuari selecciona aquest botó, s'atura l'execució i es comprova si s'ha capturat almenys 30 fotogrames i si dintre d'aquests fotogrames està inclosa la *main RGB image*. En cas afirmatiu, s'envia a l'API; en cas contrari, es descarten totes les dades obtingudes.

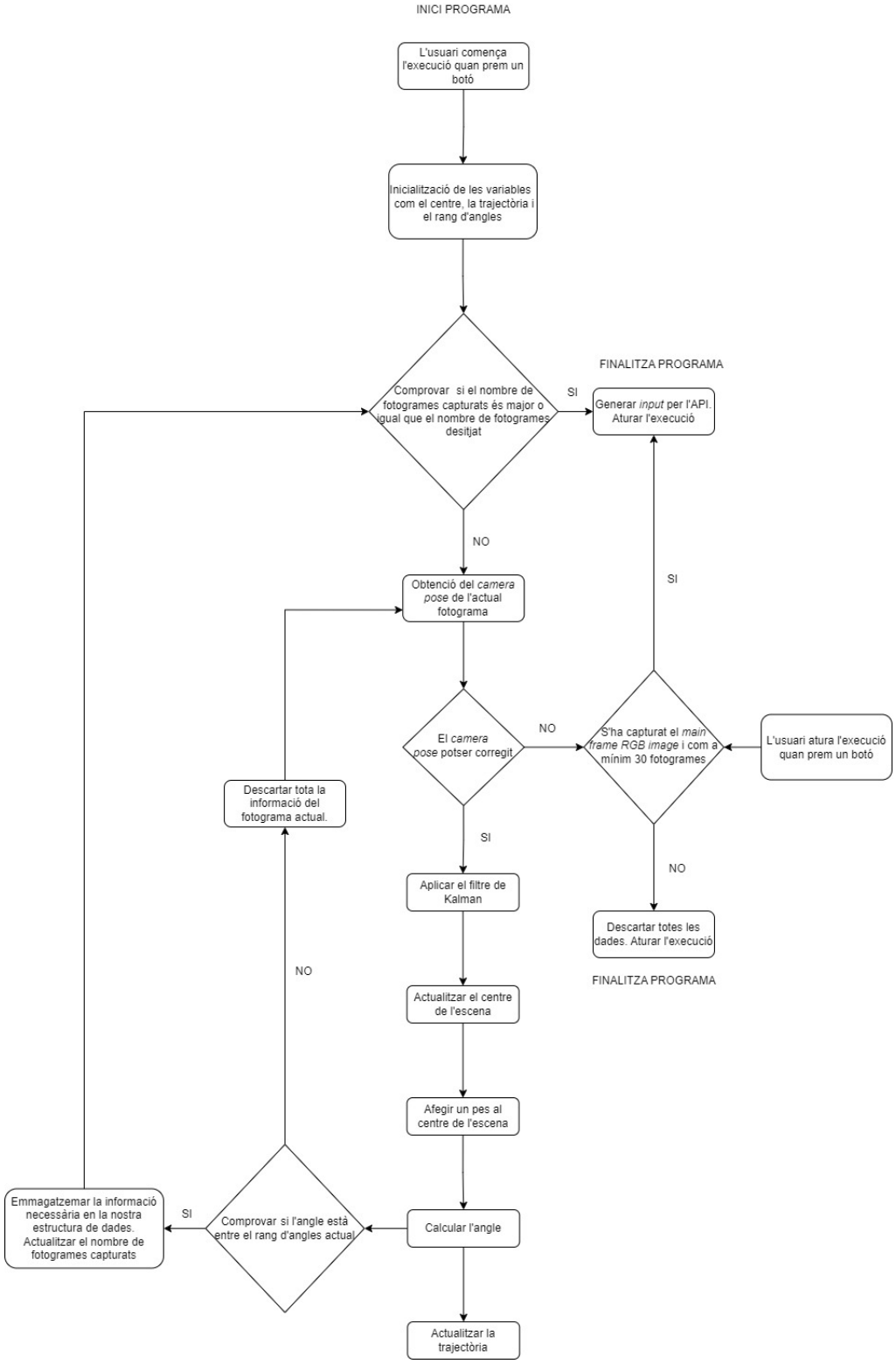

FIGURA 2.1: Imatge que mostra el diagrama de flux del procés de captura.

2.2 Anàlisis de requisits

En aquesta secció explicarem l'anàlisi de requisits en diferents nivells:

- En el context del desenvolupament de *software* és important recalcar que és necessari disposar d'un dispositiu *Android* amb unes característiques específiques per tal de dur a terme les proves, ja que, no qualsevol dispositiu *Android* serveix. Cal remarcar que s'ha utilitzat una llibreria anomenada ArCore. Aquesta és la plataforma de Google per a crear experiències de realitat augmentada i amb l'ús de diferents API, ARCore permet que el teu telèfon percebi el seu entorn, comprengui el món i interactuï amb la informació.

  ARCore usa tres capacitats clau per a integrar contingut virtual en el món real tal com es veu a través de la cambra del telèfon:

  - El seguiment de moviment permet que el telèfon comprengui i faci un seguiment de la seva posició en relació amb el món.
  - La comprensió ambiental permet que el telèfon detecti la grandària i la ubicació de tota mena de superfícies: horitzontals, verticals i angulars.
  - L'estimació de llum permet que el telèfon estimi les condicions actuals d'il·luminació de l'entorn. ArCore és capaç d'alertar si l'entorn és desfavorable en termes d'il·luminació.

- Requisits funcionals: aquest tipus de requisits estan definits de manera específica en l'apartat 1.4.

- Requisits no funcionals: Proposarem els requisits que hem identificat sobre com l'usuari hauria de realitzar la captura. S'ha de tenir en compte que aquests requisits han sigut estudiats per obtenir un equilibri entre resultat i usabilitat d'usuari. A més, explicarem quin és el rang d'angles més còmode en termes d'usabilitat. Recordem que l'objectiu principal d'aquest projecte és enviar els *inputs* necessaris a l'endpoint de *food quantity estimation* de l'API de LogMeal. Per tant, si l'usuari vol portar a cap una captura d'un aliment específic, ha de fer/saber el següent:

  - Moviments lents i controlats d'esquerra a dreta, tot dibuixant una trajectòria en forma d'arc sense perdre rastre de l'aliment, és a dir, la càmera ha d'apuntar en tot moment a l'aliment. És important recalcar que s'ha proposat el moviment d'esquerra a dreta en comptes de dreta a esquerra perquè és un moviment més natural de fer amb el dispositiu mòbil, tot i que funciona també d'esquerra a dreta.

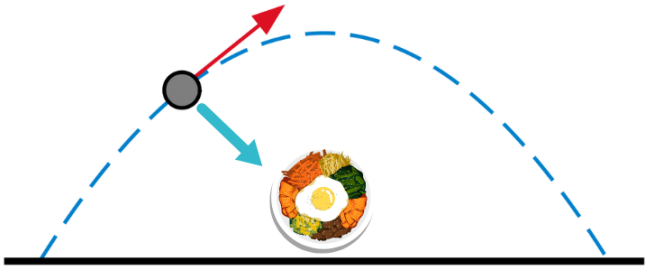

FIGURA 2.2: Imatge que mostra la trajectòria que s'espera que dibuixi l'usuari en realitzar el moviment amb el seu dispositiu, on la bola de color gris (la bola és la càmera) indica el punt d'inici. S'observa que la bola de color gris fosc està situada a l'esquerra i la fletxa de color vermell indica que el moviment del dispositiu s'ha de fer cap a la dreta. Cal destacar que el menjar aniria situat al centre de l'escena tal com es mostra amb el plat de menjar i la càmera hauria d'apuntar cap al menjar tal com s'indica amb la fletxa blava.

– S'ha afegit un *delay* de 2 segons a l'inici de la captura on no es considerarà cap dada capturada tot i que l'angle calculat pertanyi al rang adequat. Després de moltes proves s'ha observat que l'algoritme en les seves primeres iteracions era molt inestable en donar el resultat de l'angle calculat, això és pel fet que es necessiten diverses iteracions per a poder obtenir un valor del centre que tingui sentit (recordem que el centre s'inicialitza amb un valor per defecte).

  Convé subratllar que *a priori* en l'àmbit d'usabilitat tot i que sembla un inconvenient molt important, s'ha pensat que es pot solucionar de manera senzilla mostrant per pantalla una icona amb un símbol que indiqui que s'està carregant i un comptador de 2 o 3 segons en què internament ja ha començat la captura. D'aquesta manera quan a l'usuari se li mostri la interfície de captura ja podrà començar a fer el moviment perquè ja s'hauran consumit els 2 segons de *delay*.

– La trajectòria en forma d'arc que realitzarà l'usuari estarà compresa entre l'angle 50 i l'angle 130. És a dir, s'abastarà en total 80 graus. A més, Maria Fernanda va estudiar que el nombre òptim de fotogrames a capturar és 50. El motiu és senzill, amb 50 fotogrames capturats cada X angles es pot obtenir una bona representació en 3D. Una pregunta que ens pot sorgir és: per què no capturem més de 50 fotogrames? Fer el vídeo amb més fotogrames suposaria un augment de la mida considerable de l'arxiu i es va decidir finalment que el procés d'enviar la seqüència de vídeo a l'API de manera eficient era més prioritari, principalment perquè el temps d'espera de l'usuari des que finalitza la captura fins que es processen les dades per l'algoritme de *food quantity estimation* sigui el més curt possible.

FIGURA 2.3: Imatge que mostra el rang de moviment a realizar per l'usuari.

- Requisits d'usuari: L'usuari un cop seleccioni el mode de captura per iniciar-ne una, haurà d'esperar uns 2 o 3 segons en què es mostrarà una interfície d'espera tot mostrant un petit tutorial amb tota la informació necessària explicada anteriorment de com realitzar el moviment de manera correcta. La interfície d'usuari, tot i que no s'ha arribat a fer, es mostrarà una interfície conceptual de la idea que es tenia en la secció 2.4. Resumint se li proposarà a l'usuari una interfície senzilla d'utilitzar i molt visual, a més, d'un tutorial detallat de com fer la captura pas a pas.

D'altra banda, cal destacar que si volem capturar 50 fotogrames, pot passar que l'usuari no aconsegueixi capturar-los tots, per termes d'usabilitat es va considerar que tot i que no s'obtingués un resultat òptim, si hi havia un mínim de 30 fotogrames capturats on estigués inclosa la imatge principal (que és la més propera a l'angle de 90 graus) es donés com a vàlida la captura. A més, s'ha de tenir en compte que internament si l'usuari realitza la captura completa, no es capturen 50 fotogrames sinó 100, això és degut al mateix motiu que abans, termes d'usabilitat.

Finalment, és important pensar i posar-se en la situació de l'usuari en tot moment, i més quan estem parlant que segurament farà la captura quan estigui a punt de menjar, en altres paraules, hem d'aconseguir per totes les vies possibles que l'usuari faci només una captura i no hagi de repetir el procediment, és per això que s'han afegit totes aquestes eines extres per millorar la usabilitat.

2.3 Diagrama de classe

En la següent secció parlarem del diagrama de classes del codi implementat.

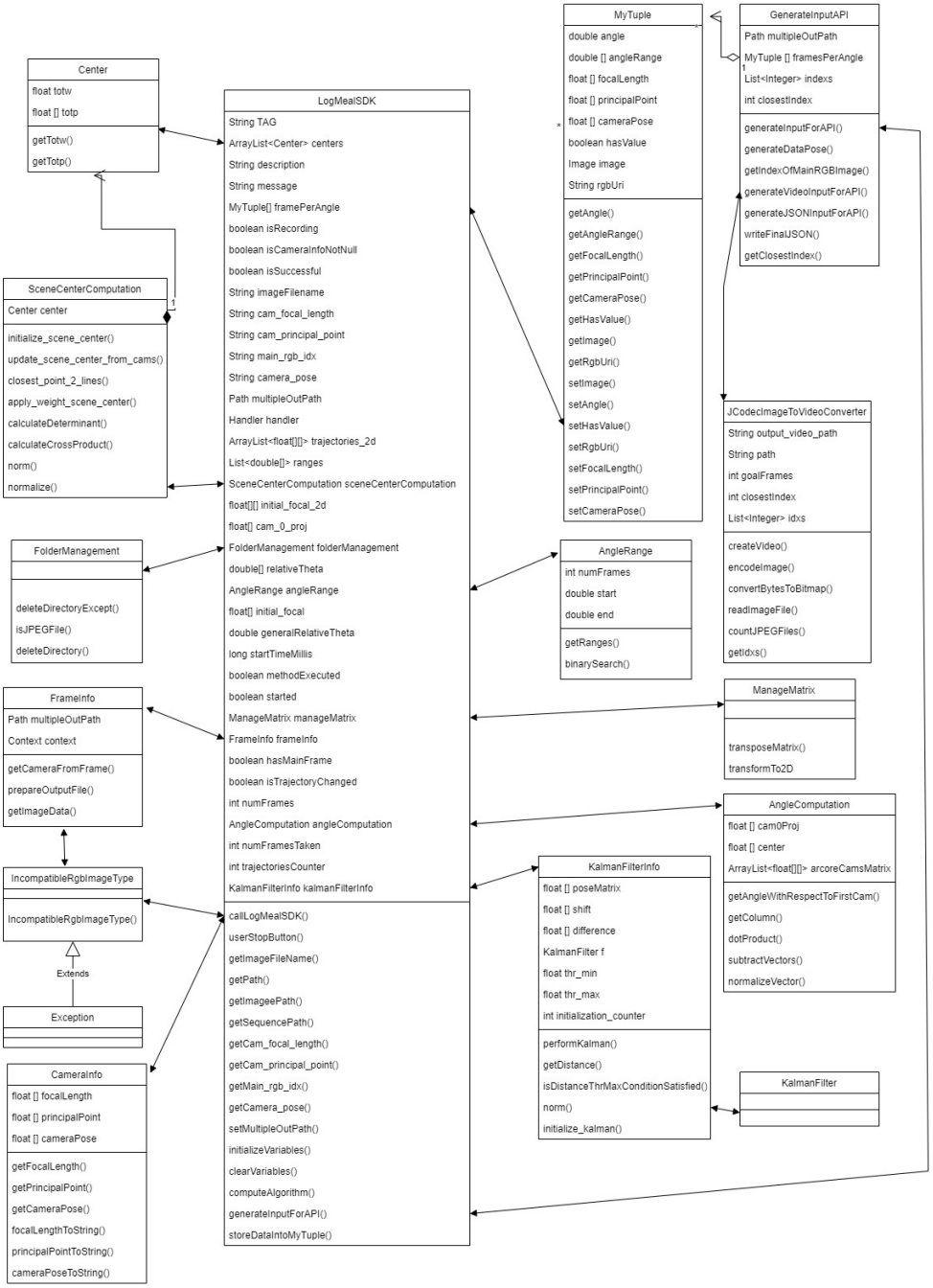

FIGURA 2.4: Imatge que mostra el diagrama de classe del codi.

En primer lloc, cal comentar que en el codi apareix una classe anomenada AdvancedIntegrationExampleActivity i és la classe que hauria d'implementar l'usuari que vol utilitzar el SDK, per tant, no la posarem en el diagrama de classe. No obstant això, més endavant explicarem com un usuari pot treballar amb aquest SDK. Totes les classes que apareixen en aquest diagrama de classe són totes les classes implementades necessàries per al funcionament correcte del SDK. A continuació explicarem de manera resumida quina és la funció de cada classe, les implementacions dels algoritmes les explicarem en la secció dedicada a implementació:

- AdvancedIntegrationExampleActivity o classe que hauria d'implementar l'usuari del SDK: Aquesta classe és on es fan les crides a totes les classes del SDK per tal de poder generar els arxius necessaris requerits per l'API. El mètode *onDrawFrame()* conté eines que ens facilita *ArCore* per poder començar a realitzar la captura. Per exemple, *Session, Frame, Camera, Anchor*. Gràcies a aquesta llibreria usant *Session* es pot controlar l'estat i el cicle de vida de la sessió actual, a més, *Frame* ens dona tota la informació del frame actual. *Anchor* ens ajuda a obtenir la localització i orientació del món real i *Camera* ens dona tota la informació sobre la càmera que és utilitzada per capturar les imatges. És important recalcar que se suposa que tota aquesta informació que acabem d'explicar sobre ArCore ja la coneix l'usuari final del SDK. En aquesta classe es farà la crida al mètode de callLogMealSDK() de la classe LogMealSDK.
- D'altra banda, LogMealSDK és una classe del SDK que conté la lògica per connectar els diferents algoritmes implementats en la resta de classes del SDK. En particular s'encarrega de:
  - Inicialitzar totes les variables: el centre, la trajectòria, la nostra estructura de dades anomenada MyTuple (*tuple* en català, tupla, vol dir una llista ordenada i finita d'elements).
  - Inicialitzar i aplicar el filtre de Kalman implementats en la classe KalmanFilterInfo
  - Actualitzar el centre de l'escena i la trajectòria, i calcular el *weighted center* implementats en la classe SceneCenterComputation.
  - Calcular l'angle entre el centre i la primera càmera detectada en la captura usant la classe AngleComputation.
  - Emmagatzemar les dades pertinents en l'estructura cridant al mètode storeDataIntoMyTuple().
  - Comprovar les condicions d'aturada del procés, explicades en el diagrama de flux en la secció 2.1
- La classe Center conté dos atributs: *totw* i *totp*, pes per cada parell de càmeres i punt que defineix el centre de l'escena, respectivament. El *totp* és un vector de la forma [x,y,z].
- La classe SceneCenterComputation conté els mètodes per fer tots els càlculs relacionats amb el centre. La classe Center i aquesta tenen una relació de composició perquè l'existència d'un objecte SceneCenterComputation depèn directament de l'existència d'un objecte Center.

- La classe FolderManagement conté tota la informació necessària per eliminar fitxers i directoris. Necessària per a quan vulguem eliminar totes les imatges capturades, excepte la *main RGB image*, un cop hàgim construït la seqüència de vídeo a partir d'aquestes.
- La classe FrameInfo conté les eines per retornar informació que s'obté a partir del *Frame* com poden ser el *focal length* i el *principal point* importants per la creació de l'arxiu en format JSON que enviarem a l'API. A més, conté un mètode que permet emmagatzemar les imatges que es van capturant en una carpeta específica.
- La classe CameraInfo conté informació de la càmera.
- La nostra estructura de dades que com hem dit anteriorment s'anomena *MyTuple* està definida en la classe *MyTuple* i conté els següents atributs entre altres: l'angle del fotograma capturat, el rang d'angle on s'ha capturat la imatge, el *focal length* i el *principal point* de la càmera, el *camera pose*...
- La classe AngleRange defineix els rangs d'angles de la captura. Per condicions d'usabilitat s'ha establert el següent criteri: es parteix de l'angle 50 fins a l'angle 130, és a dir, l'usuari ha de fer un moviment de 80 angles per tal de realitzar la captura ideal que s'espera que es faci. A més, conté un mètode *binary search* que es va implementar com a millora per abastir el cas en el qual l'usuari faci un moviment més ràpid de l'habitual i sense voler salti al següent angle. La idea era que per tal que no tornés a desplaçar el dispositiu fins a arribar al rang d'angle saltat que faci mitjançant la implementació d'un *binary search* una cerca intel·ligent a l'hora de classificar els angles per rangs. Finalment, es va optar per no implementar-ho, ja que feia més lenta l'execució, i vam considerar que educant i fent ús d'un bon tutorial explicatiu l'usuari hauria de saber fer la captura tal com la volem. Recordem que hem explicat com fer la captura en la secció 2.2.
- La classe KalmanFilterInfo conté tota la informació relacionada amb la implementació del filtre de Kalman.
- La classe GenerateInputAPI s'encarrega de la gestió en la generació dels arxius requerits per l'API.
- La implementació per la generació de l'arxiu vídeo en Java ha sigut una tasca més complicada de l'esperat. Després de l'ús de moltes llibreries com pot ser *opencv* o *frameworks* com *ffmpeg*, i altres intents d'implementacions fallits es va optar per la implementació d'una llibreria anomenada *JCodec*. Tot i que, és una llibreria que importem s'ha hagut de crear tota la lògica per formar la seqüència de vídeo. Això ha estat implementat en la classe JCodecImageToVideoConverter.
- ManageMatrix és una classe creada per efectuar operacions amb vectors com poden ser la transposada o passar d'un vector en una dimensió a una matriu 4x4.
- La classe AngleComputation conté el mètode principal per calcular l'angle a partir del centre de l'escena i de la primera càmera que es detecta un cop s'inicia el procés de captura. Cal fer un incís, i és que no estem parlant del

primer fotograma capturat en cert angle que satisfà les condicions que estigui comprès en el rang d'angles actual, estem parlant del primer fotograma quan es comença el procés d'execució.

Recordem que el nostre objectiu és crear l'arxiu JSON amb la informació necessària requerida per l'API, la generació de la seqüència de vídeo a partir de les imatges capturades cada cert rang d'angles i l'obtenció de la imatge principal de la captura (imatge capturada més propera a l'angle de 90 graus). Sabent això podem observar que amb aquesta estructura de classes tenim les eines necessàries per a aconseguir l'objectiu d'aquest projecte. Cal recalcar que només hem donat unes pinzellades sobre què fa cada classe, parlarem de la implementació dels algoritmes detallada més endavant.

2.4 Interfície d'usuari

En aquesta secció explicarem tant la interfície genèrica usada per fer les proves del SDK com la interfície conceptual que s'hagués fet amb més temps disponible. En primer lloc, com que és necessari l'ús de la llibreria ArCore hem implementat una interfície senzilla per informar a l'usuari si té instal·lat l'*apk* d'ArCore i si el dispositiu suporta les eines d'ArCore que fem servir.

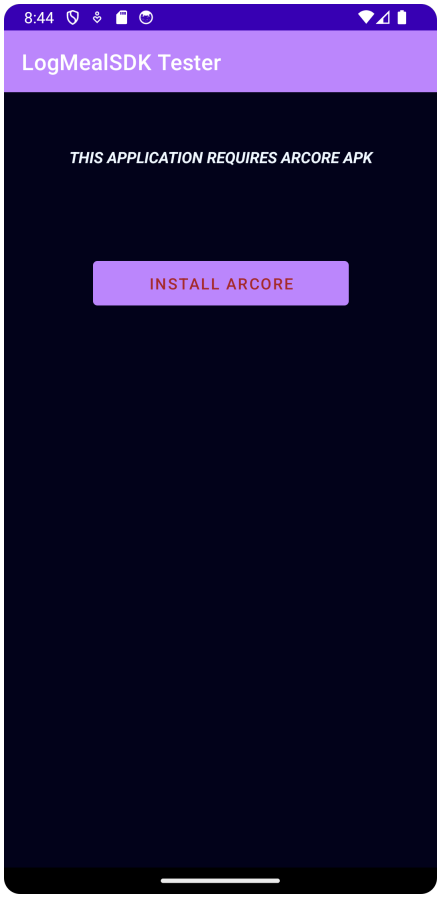

FIGURA 2.5: Imatge que mostra la interfície d'usuari en el cas que ArCore no estigui instal·lat en el dispositiu.

En la següent imatge 2.6 es visualitza la interfície que es mostra un cop executem el nostre programa en local des d'Android Studio. Com es pot veure surt un missatge de *DEPTH API ENABLED*, això vol dir que ArCore està instal·lat i que es pot utilitzar en aquest dispositiu. Cal recalcar que no tots els dispositius poden fer servir ArCore (tot i tenir-lo instal·lat), ja que, es requereixen un mínim d'especificacions en l'àmbit de maquinari per poder fer servir aquest servei. En la pàgina web d'ArCore es poden visualitzar els dispositius compatibles d'ArCore.[1]

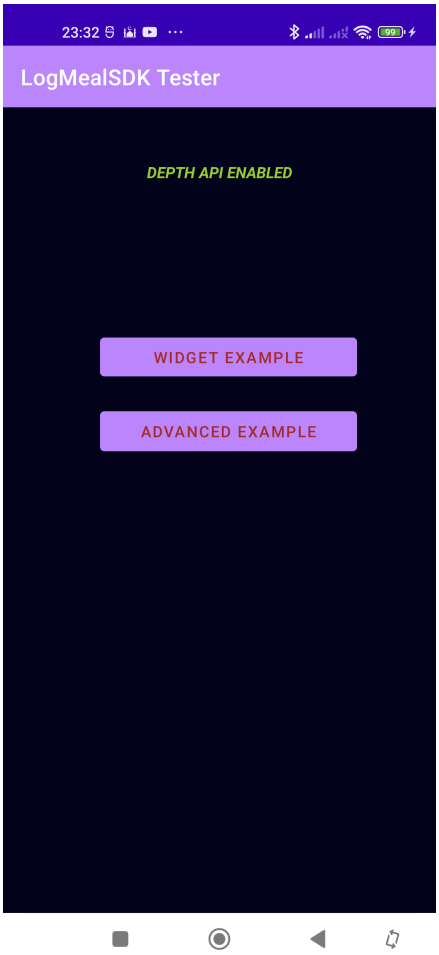

FIGURA 2.6: Imatge que mostra la interfície d'usuari en el cas que ArCore pugui ser utilitzat.

En la següent imatge 2.7 es mostra la interfície que hem usat per fer totes les proves del SDK per generar els arxius. Com es pot observar en la part superior on està la imatge d'una poma és on es visualitza la càmera en temps real. Després tenim tres botons i finalment tot i que es mostri com que està buit la part inferior de la interfície un cop s'inicia la captura mostra informació sobre: l'angle actual, el rang d'angle actual i la ubicació de la carpeta on s'estan emmagatzemant els arxius.

[1] En el següent enllaç es pot visualitzar el llistat `https://developers.google.com/ar/devices?hl=es-419`

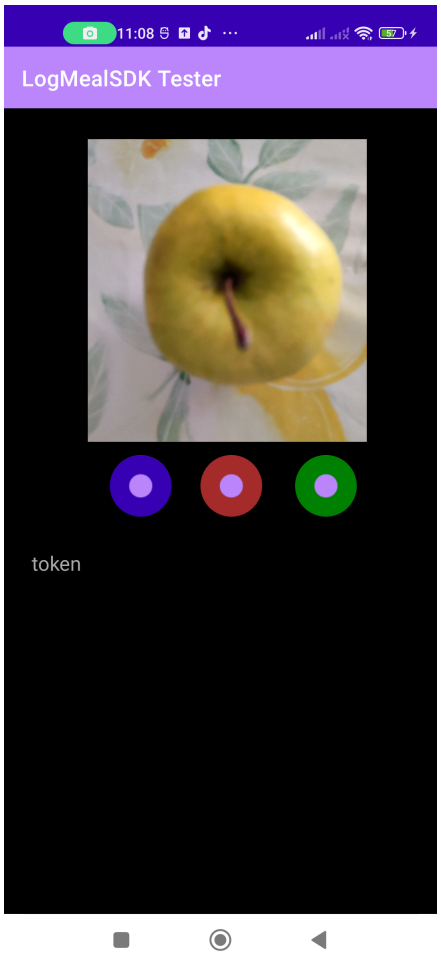

FIGURA 2.7: Imatge que mostra la interfície que hem usat per al desenvolupament.

2.4.1 Interfície d'usuari conceptual

A continuació, explicarem com hem pensat que hauria de ser la interfície d'usuari de l'aplicació per tal que l'usuari estigui informat tota l'estona de quin és l'estat actual de la captura.[2]

Com podem observar en la figura 2.8, la idea seria incorporar aquesta mena de cercle que utilitza una aplicació anomenada *MagiScan*, però en comptes de ser 360 graus, la idea seria incorporar només un arc comprés entre l'angle 50 i l'angle 130, tal com es mostra en la figura 2.3 de la secció 2.2. A més, vam considerar que un punt important és que el color dels rangs d'angles que l'usuari no pogués capturar (en la imatge és mostra en color gris) mai haurien de ser de color vermell perquè l'usuari pot interpretar que ha comès errors. Cal destacar que com vam comentar anteriorment en la secció 2.2 vam pensar que era important incorporar una interfície amb un cronòmetre d'uns 2 segons en el moment que l'usuari seleccionés el botó per iniciar captura per tal d'obtenir valors més estables del centre i així optimitzar les primeres iteracions de l'execució. Per tant, la interfície hauria de ser una pantalla amb un petit cronòmetre, que a més mostrés informació de quins passos s'haurien de seguir per tal de realitzar una bona captura.

[2]MagiScan3D és la pàgina de la qual ens hem inspirat `https://magiscan.app/`

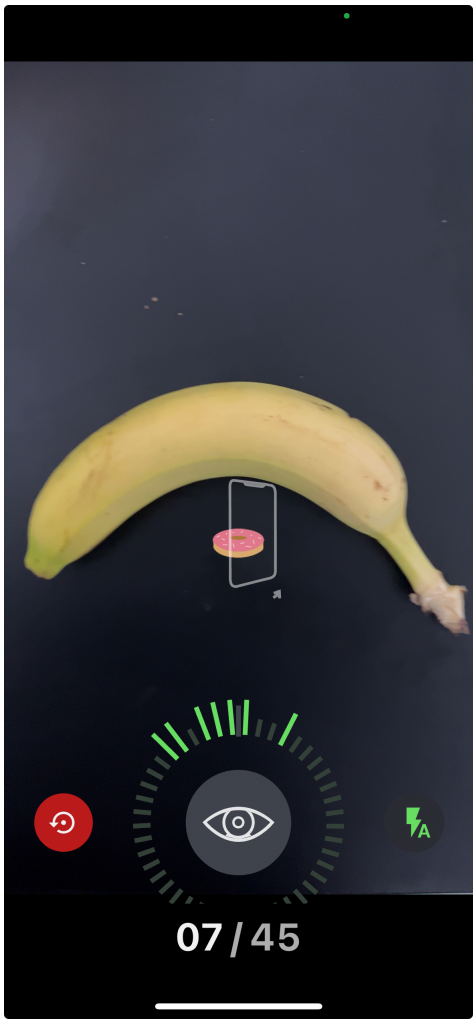

FIGURA 2.8: Imatge que mostra de manera conceptual la idea de la interfície de captura que es volia implementar. Font: App de MagiScan.

D'altra banda, un cop s'hagi realitzat la captura de manera correcta l'usuari hauria de ser desplaçat a una interfície on podria visualitzar les dades genèriques de l'aliment que acaba de capturar i un missatge que s'està processant la informació, ja que, l'*endpoint* de *food quantity estimatiom* implementat per LogMeal tarda uns 30 segons aproximadament en el servidor de proves. Convé subratllar que en el servidor de producció triga uns 8-10 segons. Un cop finalitzat el procés de detecció de l'*endpoint* l'usuari ja podria visualitzar les quantitats reals de l'aliment capturat. A més, s'hauria afegit un botó a l'esquerra per reiniciar el procés en cas que l'usuari volgués començar de nou a fer la captura.

Finalment, cal destacar que aquesta interfície conceptual s'hagués complementat amb missatges per avisar si l'usuari està realitzant els moviments de manera ràpida o no i per informar de manera dinàmica quan la captura ja satisfà els mínims vàlids per poder ser processada per l'API. Per exemple, un missatge que quan el programa detecti que ja n'hi ha 30 fotogrames capturats i la *main RGB image* doncs que avisi a l'usuari que ja se satisfan els mínims necessaris per enviar la informació a l'API.

Capítol 3

Implementació

En aquest capítol parlarem sobre la implementació dels algoritmes realitzats en el projecte.

3.1 Obtenició de les dades del frame

Per acomplir aquesta tasca com s'ha comentat anteriorment en el projecte ens hem ajudat de la llibreria d'ArCore. Cal destacar que es necessiten les següents dades: el *camera pose*, la imatge RGB i la informació de la càmera. S'han de seguir els següents passos:

- En primer lloc, s'ha de crear una *Session* per gestionar l'estat del sistema en realitat augmentada i el cicle de vida d'aquest. La realitat augmentada és el terme que es fa servir per descriure el conjunt de tecnologies que permeten que un usuari visualitzi part del món real a través d'un dispositiu tecnològic amb informació gràfica afegida per aquest.
- A continuació, mitjançant el *Session* anterior es pot obtenir el fotograma actual usant *Frame*.
- D'una manera similar s'aconsegueix la càmera actual. Partint del *Frame* fet servir a l'apartat anterior es pot aconseguir la *Camera*.
- Un cop tenim aquestes dades ja es pot obtenir el *camera pose* fent ús de *Anchor*. Aquest descriu una ubicació fixa i una orientació en el món real. Per romandre en una ubicació fixa de l'espai físic, la descripció numèrica d'aquesta posició s'actualitzarà a mesura que millori la comprensió de l'espai d'ARCore.
- Finalment, per aconseguir l'objecte imatge i la informació de la càmera com pot ser el *principal point* o el *focal length* s'ha de fer servir el *Frame* obtingut en l'apartat anterior.

D'aquesta manera s'obtenen dades del món real d'una manera molt simplificada.

Com es pot observar en la següent imatge el punt principal o *principal point* és el punt del pla de la imatge sobre el qual es projecta el centre de la perspectiva i la distància focal o *focal length* és la distància mesurada en mil·límetres, entre el centre òptic de la lent i el sensor de la càmera, on es registra la informació de la llum. Més informació a: Baeldung, 2023, Principal point and focal length

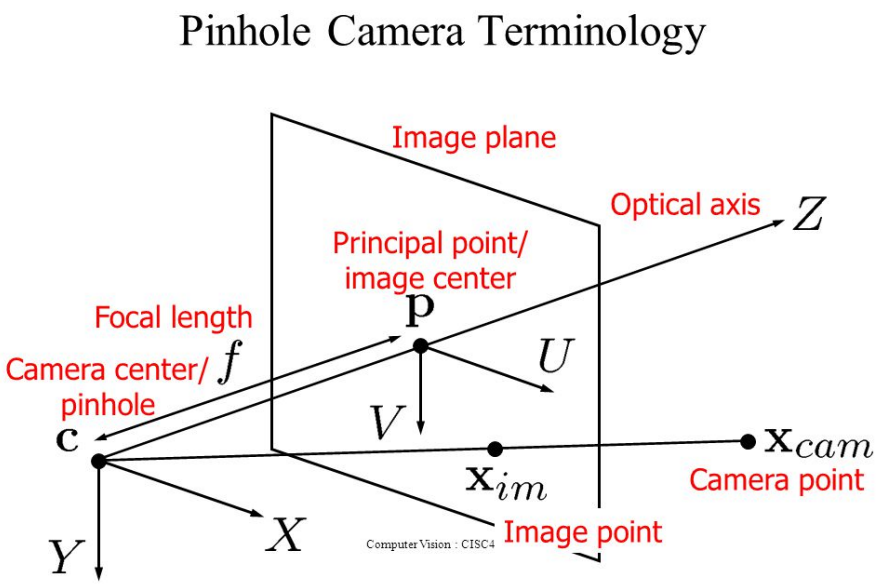

FIGURA 3.1: Imatge que mostra entre altres paràmetres de la càmera, el *principal point* i el *focal length*. Font: zeng, 2017, Scene Representation

3.2 Posició de la càmera

En aquesta secció explicarem els detalls importants sobre com tractem el *camera pose* un cop l'hem obtingut usant *Anchor*. Un cop s'aconsegueix el càmera *pose*, s'emmagatzema en un vector d'una dimensió de mida 16. Aquest vector posteriorment es transforma a una matriu 4x4. N'és un bon exemple:

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

És important observar que la posició de la càmera en el món real equival en la matriu als valors 'd', 'h' i 'l', és a dir, [x,y,z] = [d,h,l] on 'd' és el valor de la matriu situat en la quarta columna de la primera fila, 'h' és el valor situat en la quarta columna de la segona fila i 'l' és el situat en la quarta columna de la tercera fila de la matriu.

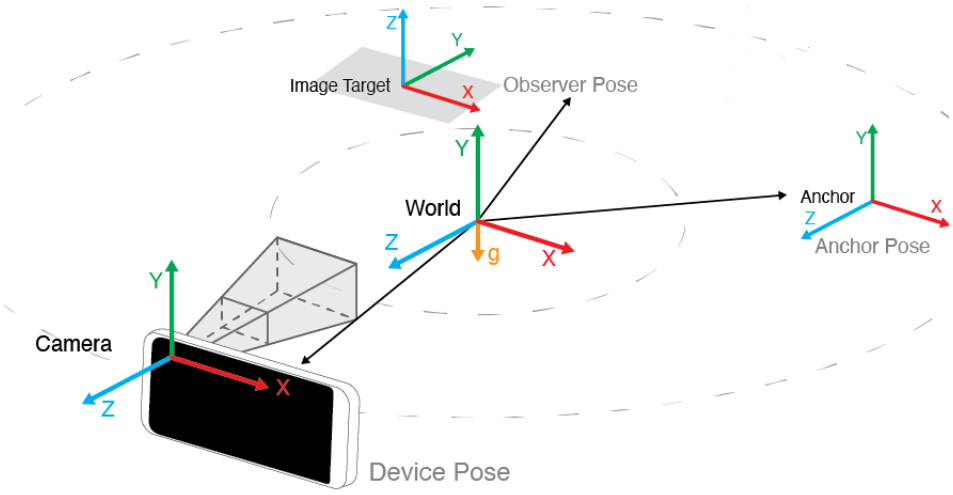

FIGURA 3.2: Imatge que mostra una visualització d'un dispositiu que apunta a un objecte del món real i com es relaciona amb *Anchor*. Font: Vuforia, 2023, Spatial Frame of Reference for Unity

3.3 Filtre de Kalman

3.3.1 Context

El filtre de Kalman és un algorisme desenvolupat per Rudolf E. Kalman en 1960 que serveix per a poder identificar l'estat ocult, en un principi no mesurable, d'un sistema dinàmic lineal quan aquest està sotmès a soroll blanc additiu. Més informació a: Wikipedia, 2023, Filtre de Kalman.

Un avantatge que presenta aquest filtre respecte d'altres filtres similars és que el guany K de realimentació de l'error és capaç de triar-lo de manera òptima quan es coneixen les variàncies dels sorolls que afecten el sistema. El filtre de Kalman normal és un algorisme recursiu, que pot ser executat en temps real usant únicament els mesuraments d'entrada actuals, l'estat calculat prèviament i la seva matriu d'incertesa, i no requereix cap altra informació addicional.

El filtre Kalman s'utilitzarà per a identificar i corregir els errors de seguiment i moviment, ja que, prediu la posició actual de la càmera basada en el model kalman que explicarem més endavant i la posició prèvia de la càmera. Després, compara la predicció amb la posició de la càmera arcore i decideix si la posició de la càmera arcore és correcta. Si la posició de la càmera arcore és correcta, s'actualitza el model kalman. Aquest procediment es repeteix cada vegada que adquirim un nou fotograma.

Per tant, es realitzarà la implementació d'un filtre de Kalman per estimar l'estat x_k d'un procés controlat en temps discret que es regeix per l'equació de diferència

estocàstica lineal on:

$$x_k = A * x_{k-1} + B * u_{k-1} + w_{k-1} \tag{3.1}$$

$$z_k = H * x_k + v_k \tag{3.2}$$

on x_k és l'estat vertader i z_k la seva estimació en el temps k. Les variables aleatòries wk i vk representen el procés i el soroll de mesura i s'assumeix que són independents l'una de l'altra i es distribueixen amb probabilitat normal (*white noise* o soroll blanc). És a dir:

- w_k es el soroll blanc on el valor de la mitjana és 0 i en l'instant k té variança Q_k
- v_k es el soroll blanc on el valor de la mitjana és 0 i en l'instant k té variança R_k

Per tant, s'observa que el filtre de Kalman permet estimar l'estat x_k a partir de les mesures anteriors de u_{k-i}, z_{k-i}, Q_{k-i}, R_{k-i} i les estimacions anteriors de x_{k-i}.

El cicle del filtre Kalman implica els següents passos:

- Predicció: projecta l'estimació de l'estat actual en el temps
- Correcció: ajusta l'estimació projectada amb una mesura real

FIGURA 3.3: Imatge d'una gràfica on es mostra un exemple pràctic de l'aplicació del filtre de Kalman. Suposem que volem estimar l'altura d'un edifici fent ús d'un altímetre imprecís. L'alçada de l'edifici és de 50 metres. La gràfica en l'eix de les ordenades mostra l'alçada en metres i en l'eix de les abscisses mostra el nombre d'iteracions. En color verd es mostra el valor real, en color vermell el valor estimat i en color blau el valor de mesura. Es pot observar com l'error d'estimació va disminuint a mesura que augmenten les iteracions, recordem que aquest error es calcula com la diferència entre els valors reals (línia verda) i els valors estimats (línia vermella). Font: Becker, 2023, Kalman Filter

3.3.2 Paràmetres

El filtre Kalman s'inicia amb un *ProcessModel* i un *MeasurementModel*, que contenen les corresponents matrius de transformació i covariància del soroll. A més, el *ProcessModel* defineix el model de dinàmica de processos i *MeasurementModel* defineix el model de mesura. Els noms dels paràmetres utilitzats en els models respectius corresponen als següents noms utilitzats habitualment en la literatura matemàtica. A continuació detallarem quins són els paràmetres usats pel filtre de Kalman:

- A - matriu de transició d'estats
- B - controla la matriu d'entrada
- H - matriu de mesura
- Q - matriu de covariància del soroll del procés
- R - matriu de covariància del soroll de mesura
- P - matriu de covariància d'error

3.3.3 Inicialització

En primer lloc, convé fer ressaltar que es crearà el filtre de Kalman a partir del *ProcessModel* i del *MeasurementModel*. Passos a seguir per inicialitzar el filtre de Kalman:

- Primerament definir *Kalman Filter* com una matriu 9x3. On el valor 9 correspon a la mida del vector d'estat, mentre que el valor 3 correspon a la mida del vector de mesura.
- Inicialitzar un vector d'estat de mida 9 que anomenem *initialState* de la següent manera:
$$[x, 0, 0, y, 0, 0, z, 0, 0] \tag{3.3}$$
on x,y,z són els valors del món real capturats de la primera càmera.
- A continuació, s'ha d'inicialitzar la matriu de transició d'estats 'A' amb uns valors predeterminats.
- Després s'ha de definir la matriu de mesura 'H' amb els valors estàndard.
- Inicialitzar la matriu de covariància 'P' com una matriu identitat multiplicada per un escalar.
- Posteriorment, inicialitzar també la matriu de covariància 'R' del soroll de mesura com una matriu identitat multiplicada per un escalar.
- Definir la matriu de covariància del soroll del procés 'Q'.

Un cop tenim els paràmetres de Kalman inicialitzats hem de definir el *ProcessModel* i el *MeasurementModel*. El primer depèn de: la matriu 'A' de transició d'estats, la matriu de 'Q' de covariància del soroll del procés, el vector *initialState* i de la matriu 'P' de covariància d'error; i el segon depèn de la matriu 'H' de mesura i de la matriu 'R' de covariància del soroll de mesura.

Finalment, un cop tenim totes aquestes eines definides, ja es pot crear el filtre de Kalman fent servir com s'ha comentat anteriorment el *ProcessModel* i el *MeasurementModel*.

3.3.4 Predicció

En aquest subapartat explicarem el model de predicció de Kalman. Cal tornar a dir que quan es parla de predicció es fa referència a l'objectiu d'estimar el següent estat.

Convé subratllar que un cop es realitza la predicció del següent estat és necessari calcular la distància entre l'estat predit i l'estat actual per tal de decidir si aquest error es pot corregir o no. Aquesta distància es defineix com la norma de la diferència entre el vector de l'estat estimat i el vector de l'estat actual, on la norma es defineix com l'arrel quadrada de la suma dels quadrats de cada element del vector. Un cop s'obté aquesta distància es compara amb uns *thresholds* prèviament definits per tal de decidir si aquest valor es pot corregir o no.

- Si es pot corregir, ens desplacem al següent estat que és l'estat de correcció.
- Si no es pot corregir, vol dir que l'usuari ha fet un moviment amb el dispositiu que com a conseqüència desemboca en un canvi de trajectòria i, per tant, el procés de captura s'hauria d'aturar.

3.3.5 Correcció

La correcció en el filtre de Kalman es realitza sobre els punts 'x','y' i 'z' del *camera pose* del fotograma actual. Recordem que la correcció es du a terme si la diferència entre l'estat estimat i l'estat actual es troba entre uns valors o *thresholds* arbitràriament definits. Un cop es fa la correcció el valor de 'x','y' i 'z' del *camera pose* es veurà modificat. És important recalcar que aquesta correcció es realitza tant si és un error de moviment del dispositiu de l'usuari com si és un error intern que *Android* genera per defecte en la interpretació del *camera pose* actual. [1]

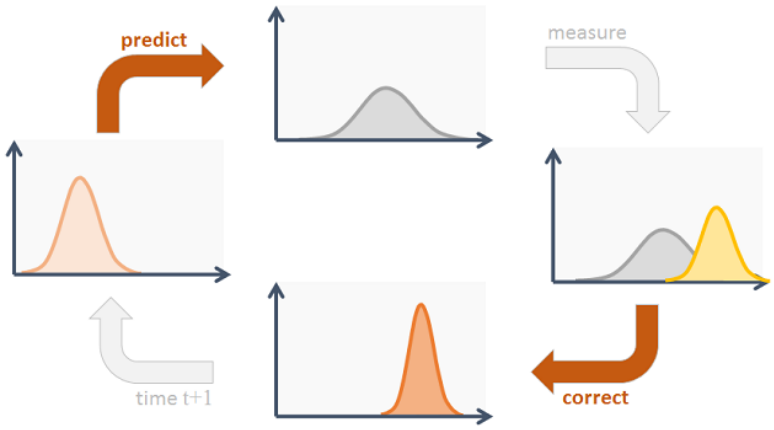

FIGURA 3.4: Imatge que mostra el procés d'iteració del filtre de Kalman. Font: Becker, 2023, Kalman Filter

[1] Més informació a: https://es.wikipedia.org/wiki/Filtro_de_Kalman

3.4 Centre de l'escena

3.4.1 Context

Obtenir el centre de l'escena és útil per definir l'angle de les càmeres respecte al centre. Per a cada fotograma de la càmera, podem definir una línia que travessa tant el centre òptic com el punt principal, anomenat eix òptic, que es pot definir a partir de la tercera i segona columna de la matriu de la càmera.

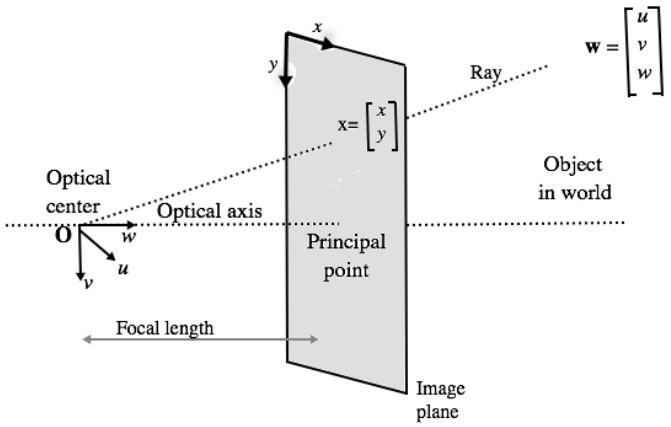

FIGURA 3.5: Imatge que mostra el centre òptic, l'eix òptic, la distància focal, el punt principal i el pla que defineix la imatge. Font: Azarcoya-Cabiedes, 2014, Centre òptic

Convé subratllar que el centre està definit per dos atributs com s'ha comentat anteriorment en la secció 2.3: *totw* i *totp*, pes per cada parell de càmeres i punt que defineix el centre de l'escena, respectivament. El *totp* és un vector de la forma [x,y,z].

Si comparem parells de línies (línies obtingudes a partir de dos marcs), podem trobar la intersecció d'eixos òptics (és improbable trobar una intersecció perfecta, de manera que estem trobant el punt més proper en el seu lloc). El centre de l'escena s'aconsegueix com la mitjana ponderada de les interseccions, on el pes és 0 si dues línies són paral·leles.

La imatge següent mostra tota la projecció de tots els eixos òptics i com convergeixen a la mateixa regió. Aquesta imatge és un bon exemple per visualitzar el centre de l'escena en una captura d'imatges.

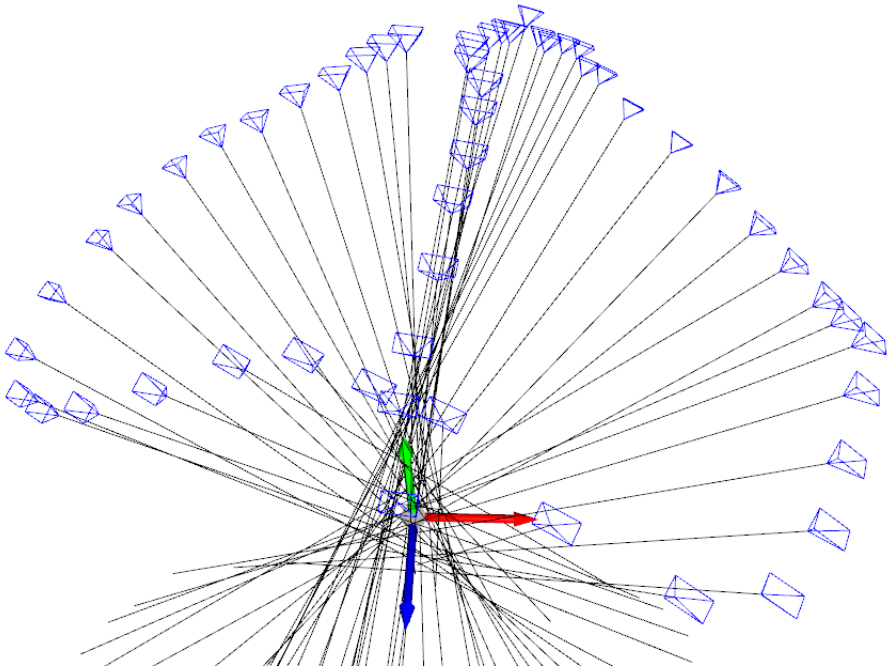

FIGURA 3.6: Imatge obtinguda amb *Python* que mostra la convergència cap al centre de l'escena dels diferents *camera pose* obtinguts en una captura.

3.4.2 Inicialització

Per aconseguir el que es mostra en la figura 3.6 s'ha de cridar a l'inici de la sessió a una funció que inicialitza el centre de l'escena amb valors per defecte:

- El valor de *totw* es defineix com 0.
- El vector *totp* es defineix com:

$$[0, 0, 0] \tag{3.4}$$

3.4.3 Actualització del centre de l'escena

La funció per actualitzar el centre es crida cada vegada que adquirim un nou fotograma. Compara cada nova matriu de càmera amb totes les matrius de càmeres anteriors i actualitza el centre i el pes de manera adient. A continuació mostrarem el *pseudo-codi* del mètode per actualitzar el centre de l'escena:

Algorithm 1: Actualització del centre de l'escena

Data: trajectory, center
Result: totw, totp

```
/* trajectory is a list of all the cameras                    */
/* Initialize variables                                       */
totw, totp ← center;
last_cam ← getLastElement(trajectory);
lc ← extractFirst3Rows(last_cam);
lc_col_3 ← extractColumn(lc, 3);
lc_col_2 ← extractColumn(lc, 2);
/* Loop through each camera in the trajectory                 */
for cam in trajectory do
    tc ← extractFirst3Rows(cam);
    tc_col_3 ← extractColumn(tc, 3);
    tc_col_2 ← extractColumn(tc, 2);
    p, w2 ← closestPoint2Lines(lc_col_3, lc_col_2, tc_col_3, tc_col_2);
    /* Update totals if condition is met                      */
    if w2 > 0.00001 then
        totp ← totp + p · w2;
        totw ← totw + w2;
return totw, totp
```

L'algoritme per actualitzar el centre de l'escena té com a paràmetres la trajectòria que és una estructura que emmagatzema tots els *camera pose* i el centre de l'escena actual. Aquest algoritme està dissenyat per actualitzar el centre d'una escena basada en trajectòries de càmera i el centre actual. Aquí hi ha una explicació pas a pas:

1. Inicialitza *totw* i *totp* amb el pes i la posició del centre actual.
2. Extreu informació de la darrera càmera de la trajectòria.
    - Extreu les 3 primeres files de la matriu de trajectòria per crear una matriu *lc*
    - Extreu les columnes corresponents als índexs 3 i 2 de *lc* per a crear matrius d'una sola columna, *lc_col_3* i *lc_col_2* respectivament.
3. Itera per cada càmera de la trajectòria:
    - Extreu les 3 primeres files de la càmera de la trajectòria per a crear una matriu *tc*.
    - Extreu les columnes corresponents als índexs 3 i 2 de *tc* per a crear matrius d'una sola columna, *tc_col_3* i *tc_col_2* respectivament.
    - Utilitza la funció *closestPointToLines* per trobar el punt i el pes més propers entre les línies definides per *lc_col_3*, *lc_col_2*, *tc_col_3* i *tc_col_2*.
    - Si el pes *w2* obtingut de l'anterior funció és superior a un llindar que es defineix com 0,00001, s'actualitza *totp* i *totq* utilitzant el punt i el pes obtinguts.
4. Es crea un objecte Centre nou amb el *totw* i *totp* actualitzats.

5. Es retorna el centre actualitzat.

En resum, l'algoritme està actualitzant el centre basant-se en la informació extreta de la darrera càmera i totes les càmeres de la trajectòria. Utilitza un mètode (punt més proper entre dues línies) per trobar el punt més proper i el pes entre línies en un espai 3D, i si el pes està per sobre d'un cert llindar, actualitza la posició total i el pes del centre.

Convé destacar que la nostra funció per actualitzar el centre depèn completament de l'algorisme que cerca el punt més proper entre dues línies, per tant, és necessari analitzar que realitza aquest. A continuació mostrarem el *pseudo-codi* de l'algorisme *closest point between two lines* i explicarem detalladament que realitza aquest:

3.4.4 Algoritme per calcular el punt més proper entre dues línies:

Assumeix que tens dues línies representades per equacions paramètriques:

Línia 1: $\mathbf{P}_1 = \mathbf{a} + t\mathbf{u}$
Línia 2: $\mathbf{P}_2 = \mathbf{b} + s\mathbf{v}$

On $\mathbf{a}$ i $\mathbf{b}$ són punts de les línies, $\mathbf{u}$ i $\mathbf{v}$ són vectors de direcció de les línies, i t i s són paràmetres.

1. **Estableix les equacions:**
   - $\mathbf{w} = \mathbf{a} - \mathbf{b}$
   - $\mathbf{u} \cdot \mathbf{u} = |\mathbf{u}|^2$
   - $\mathbf{u} \cdot \mathbf{v}$
   - $\mathbf{v} \cdot \mathbf{v} = |\mathbf{v}|^2$
   - $\mathbf{w} \cdot \mathbf{u}$
   - $\mathbf{w} \cdot \mathbf{v}$
2. **Calcula els paràmetres t i s:**

$$t = \frac{(\mathbf{w} \cdot \mathbf{u})(\mathbf{v} \cdot \mathbf{v}) - (\mathbf{w} \cdot \mathbf{v})(\mathbf{u} \cdot \mathbf{v})}{|\mathbf{u}|^2|\mathbf{v}|^2 - (\mathbf{u} \cdot \mathbf{v})^2}$$
$$s = \frac{(\mathbf{w} \cdot \mathbf{u})(\mathbf{u} \cdot \mathbf{v}) - (\mathbf{w} \cdot \mathbf{v})(\mathbf{u} \cdot \mathbf{u})}{|\mathbf{u}|^2|\mathbf{v}|^2 - (\mathbf{u} \cdot \mathbf{v})^2}$$

3. **Calcula els punts més propers $\mathbf{P}_{\text{més proper},1}$ i $\mathbf{P}_{\text{més proper},2}$:**

$$\mathbf{P}_{\text{més proper},1} = \mathbf{a} + t\mathbf{u}$$
$$\mathbf{P}_{\text{més proper},2} = \mathbf{b} + s\mathbf{v}$$

Ara, $\mathbf{P}_{\text{més proper},1}$ i $\mathbf{P}_{\text{més proper},2}$ són els punts més propers a la Línia 1 i Línia 2, respectivament. La distància entre aquests punts dona la distància mínima entre les dues línies.

Recorda que si les línies són paral·leles, potser no hi ha un punt més proper únic. En aquests casos, pots triar qualsevol punt d'una de les línies com a punt més

proper. Aleshores, un cop tenim clara la teoria sobre com implementar la funció es proposa:

Algorithm 2: Punt més proper entre dos línies

Data: oa, da, ob, db
Result: result, denom

`/* returns point closest to both rays of form` $o + t \cdot d$ `*/`
$aux\_da \leftarrow \text{normalize}(da)$;
$aux\_db \leftarrow \text{normalize}(db)$;
$c \leftarrow \text{calculateCrossProduct}(aux\_da, aux\_db)$;
$denom \leftarrow \text{norm}(c)^2$;
$t \leftarrow \text{subtract}(ob, oa)$;
$aux\_ta \leftarrow [t, db, c]$;
$aux\_tb \leftarrow [t, da, c]$;
$ta \leftarrow \frac{\det(\text{createRealMatrix}(aux\_ta))}{denom+1e-10}$;
$tb \leftarrow \frac{\det(\text{createRealMatrix}(aux\_tb))}{denom+1e-10}$;
if $ta > 0$ **then**
  $ta \leftarrow 0$
if $tb > 0$ **then**
  $tb \leftarrow 0$
$result \leftarrow \text{average}(oa + ta \cdot da, ob + tb \cdot db)$;
return $\{result, denom\}$

El *pseudocodi* proporcionat representa l'algorisme *closest_point_2_line* que calcula el punt més proper entre dos raigs a l'espai. Aquí hi ha una explicació de cada part:

1. Normalització:
   - Es normalitza *da* i *db* que són els respectius *lc_col_2* i *tc_col_2* que recordem que es crida en la funció d'actualitzar el centre de l'escena.
   - A continuació es calcula el producte vectorial dels vectors prèviament normalitzats.
   - El valor anomenat *denom* s'obté fent la potència quadrada del valor *c* normalitzat.
2. Càlcul de *t*:
   - El vector *t* és el resultat de restar *oa* i *ob*, on aquests valors són els respectius *lc_col_3* i *tc_col_3*.
3. Configuració de les matrius:
   - Es crea una matriu auxiliar conformada pels vectors *t*, *db* i *c*.
   - Es crea una altra matriu auxiliar conformada pels vectors *t*, *da* i *c*.
4. Càlcul del determinant:
   - Els valor *ta* i *tb* es calculen fent la divisió del determinant de les matrius creades en el pas anterior pel valor *denom* + *epsilon* calculat anteriorment. El valor *epsilon* és per evitar la divisió per zero.

5. Llindar:
    - La comprovació del llindar assegura que els valors negatius de *ta* i *tb* es tracten com a 0 per evitar
6. Retornem *result* que representa el punt més proper entre dos raigs.

L'algorisme utilitza conceptes matemàtics per trobar una solució al problema de determinar el punt més proper entre dos raigs en l'espai 3D. S'apliquen conceptes de matemàtiques vectorials, àlgebra lineal i geometria per formular i resoldre el problema.

3.4.5 Actualització del centre de l'escena

Per concloure la secció relacionada amb el centre de l'escena parlarem de la funció que aplica un pes al centre. Aquest pas calcula essencialment la mitjana ponderada del vector de posició, on cada component d'aquest es divideix pel pes total. La nostra funció retorna el punt del centre amb el pes aplicat. Aquest centre és el que es fa servir per calcular l'angle.

3.5 Computació de l'angle

Un cop tinguem el centre d'escena, assumirem que el centre es troba en un pla. Utilitzarem aquest pla i la posició de la càmera per obtenir diferents angles que s'utilitzaran per caracteritzar la posició de la càmera respecte al centre. Per aconseguir l'angle és necessari actualitzar el centre d'escena i els angles per a cada càmera després de cada adquisició de fotogrames.

3.5.1 Primera versió

En una primera versió de l'algorisme vam plantejar d'obtenir l'angle fent ús de l'azimut, l'alçada i la distància angular entre qualsevol càmera i la primera càmera. La següent imatge mostra el significat d'azimut i altitud:

No obstant això, no es va poder obtenir un resultat numèric del tot correcte amb aquesta implementació, ja que, obteníem uns salts entre angles que feien que la captura de fotogrames per rangs no es pogués dur a terme.

Per exemple, en la implementació un cop detectava que s'arribava a l'angle 78 hi havia un salt a l'angle 98. Convé subratllar que un dels objectius és capturar fotogrames per cada rang d'angles prèviament establert. Com s'ha comentat anteriorment en el projecte, el rang de trajectòria que s'espera és de l'angle 50 fins al 130 i recordem que nosaltres tot i que volem només 50 fotogrames en capturem 100 per termes d'usabilitat. Per tant, haurem de prendre un fotograma cada (130-50) / 100 angles, és a dir, cada 0.8 angles. En conseqüència, no podríem capturar fotogrames en el rang d'angles [78.00, 78.80], [78.80, 79.60], ..., [98.00, 98.80].

Com a resultat, s'ha hagut de treballar en la investigació i implementació d'un altre algorisme.

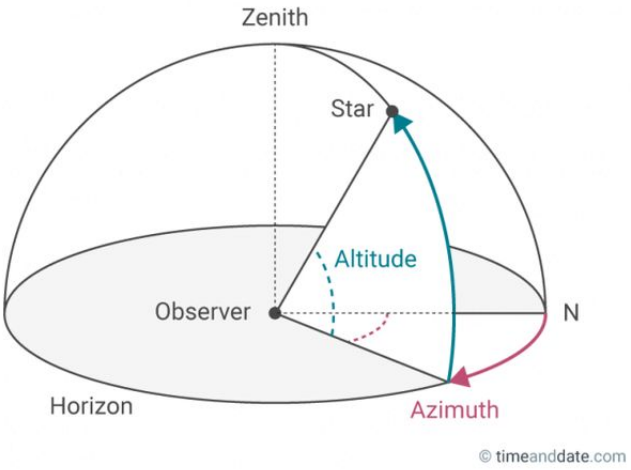

FIGURA 3.7: Imatge que mostra els paràmetres d'azimut i altitud. Font: Azarcoya-Cabiedes, 2014, The Horizontal Coordinate System

3.5.2 Segona versió

Aquest algorisme calcula els angles entre els vectors de direcció des del centre de l'escena a cada càmera del conjunt total d'aquestes, respecte al vector de direcció des del centre de l'escena a la primera càmera del primer fotograma detectat.

La següent explicació detalla el codi que defineix un mètode anomenat *getAngleWithRespectToFirstCam* on els angles es calculen basant-se en els vectors normalitzats que representen les posicions de les càmeres respecte al centre de l'escena.

1. **Inicialitza les variables:**
   - Crea un array *relativeTheta* per emmagatzemar els angles entre la primera càmera i cada càmera subseqüent.
2. **Calcula el vector per a la primera càmera:**
   - Calcula el vector *cam2CenterVector0* que representa la posició de la primera càmera respecte al centre de l'escena. Aquest vector s'obté normalitzant la resta de *cam0Proj* (projecció de la primera càmera) i *center*.
3. **Itera a través de cada càmera:**
   - Utilitza un bucle per iterar sobre cada càmera a *arcoreCamsMatrix*.
   - Per a cada càmera, obté el vector de posició respecte al centre de l'escena (*cam2CenterVector*) normalitzant la resta de la posició de la càmera en el món real i el *center*.
   - Calcula l'angle *theta* entre la primera càmera i la càmera actual utilitzant el producte escalar i l'arc cosinus.
   - Emmagatzema l'angle calculat *theta* a l'array *relativeTheta* a l'índex corresponent.
4. **Retorna el resultat:**

Algorithm 3: Càlcul de l'angle respecte a la primera càmera

Result: relativeTheta

```
/* Calculate vector for the first camera                              */
cam2CenterVector0 ← normalizeVector(subtractVectors(cam0Proj, center));
/* Iterate through each camera                                        */
for i ← 0 to arcoreCamsMatrix.size() − 1 do
    cam ← arcoreCamsMatrix.get(i);
    /* Calculate vector for the current camera                        */
    cam2CenterVectorAux ← getColumn(cam, 3);
    cam2CenterVectorAux2 ←
     [cam2CenterVectorAux[0], cam2CenterVectorAux[1], cam2CenterVectorAux[2]];

    cam2CenterVector ←
     normalizeVector(subtractVectors(cam2CenterVectorAux2, center));
    /* Calculate angle between the first camera and the current
       camera                                                         */
    θ ←
     toDegrees(cos^−1(dotProduct(cam2CenterVector0, cam2CenterVector)));
    relativeTheta[i] ← θ;
return relativeTheta
```

- Retorna el vector *relativeTheta* que conté els angles entre la primera càmera i cada càmera subseqüent.

En resum, el mètode calcula els angles relatius entre la primera càmera i cada càmera a la matriu, proporcionant informació sobre la distribució de les càmeres al voltant del centre de l'escena. El codi utilitza operacions amb vectors i funcions trigonomètriques per aconseguir aquest objectiu.

Finalment, cal destacar que aquest algorisme és sensible a les primeres iteracions del procés de captura perquè fem la captura de manera dinàmica, i a més, hem vist que depèn del centre de l'escena, i aquest en les primeres iteracions de la captura és inestable.

3.6 Generació dels fitxers

En aquesta secció explicarem com generem la seqüència de vídeo en la subsecció 3.6.1, l'arxiu JSON en 3.6.2, i la imatge principal de la captura en 3.6.3.

3.6.1 Seqüència de Vídeo

Per fer la generació de la seqüència de vídeo a partir de fotogrames s'ha hagut de fer el següent:

- En primer lloc, ubicar l'arxiu de vídeo en el mateix directori que la resta d'arxius que generem.
- Recordar que, tot i que, l'objectiu ideal és capturar 50 fotogrames, es capturen 100 per millorar els termes d'usabilitat. Es proposa capturar 100 fotogrames per la casuística següent:

  - L'usuari comença una nova captura i abans d'acabar-la, aquest finalitza el procés. Cal recalcar que l'usuari només veu que s'han de capturar 50 fotogrames. Per tant, imaginem-nos que l'usuari un cop ha capturat 15 fotogrames decideix per la raó que sigui finalitzar el procés. Tècnicament amb 15 fotogrames no es podria fer una bona representació, és per això que internament es captura el doble de fotogrames i d'aquesta manera, tot i que, l'usuari pensa que n'ha capturat 15, en realitat s'han capturat 30 (i si a més es troba la imatge principal) es podria generar els *inputs* per l'API.

- Per requisits tècnics de l'API el vídeo s'ha de generar amb les següents especificacions:

  - Resolució: 480 x 360
  - Fotogrames per segon: 60

  Cal comentar que com més fotogrames per segon menys pes presenta l'arxiu de la seqüència de vídeo perquè el vídeo tindria una durada més petita, motiu molt important per optimitzar l'enviament de dades a l'API. Cal destacar que en iteracions prèvies del procés es va considerar una resolució de 640x360 i 30 frames per segon.

- Codificar cada imatge que formarà part de la seqüència de vídeo amb la resolució desitjada.

A continuació, situem-nos en la casuística en què es capturen els 100 fotogrames. Procés per seleccionar els 50 fotogrames per realitzar el vídeo (aquest procediment és extrapolable a qualsevol nombre de fotogrames capturat):

- Recordem que si es capturen 100 imatges, els noms de les imatges van des de: image0 fins a image99. En la funció implementada se seleccionen les imatges parelles [2]. Les imatges imparelles es descarten totes exceptuant el cas en què la imatge principal sigui imparell. Convé subratllar que la imatge principal és la imatge més important de la captura, i per tant en aquest cas excepcional s'afegiria la imatge imparella i conseqüentment es descartaria la següent imatge parella.

3.6.2 Arxiu JSON

L'arxiu JSON ha de presentar els següents requisits:

- La distància focal de la càmera.
- El punt principal de la càmera.
- L'índex que representa la imatge principal capturada. Com que es capturen 50 fotogrames, l'índex ha d'estar entre 0 i 49.
- Un diccionari amb tots els *camera pose* dels fotogrames capturats.
- La versió del SDK, en aquest cas *Android*.

[2] Abús de notació: per imatges parelles entenem que són aquelles en què el dígit que apareix en el nom del fitxer és parell, pel cas imparell és el mateix

3.6.3 Imatge principal

En el procés de captura, per cada pas de la iteració, un cop s'obté el següent fotograma vàlid s'emmagatzema la imatge en memòria. En efecte, és necessari guardar en memòria tots els fotogrames vàlids per la captura perquè *a posteriori* es pugui formar la seqüència de vídeo.

Un cop format el vídeo es descarten totes les imatges menys la imatge principal. S'ha implementat un algorisme que detecta quina és la imatge principal que recordem que és aquella imatge que es troba més propera a l'angle de 90 graus.

3.7 Procediment complet

Inicialitza una trajectòria, el centre de la trajectòria i el *shift* o desplaçament de la trajectòria:

- Una trajectòria contindrà totes les posicions de la càmera per a la trajectòria actual. La primera trajectòria serà inicialitzada amb la primera postura de càmera.
- El centre s'inicia amb un punt a [0,0,0] i un pes amb valor 0
- El desplaçament és un vector que acumularà errors de seguiment, es comença com [0,0,0]

Itera sobre cada fotograma:

1. Actualitza la posició de la càmera amb errors de *tracking* o seguiment anteriors
2. Obtenir la predicció de Kalman
3. Mesura la diferència entre (1) la posició de la càmera desplaçada i (2) la predicció del filtre kalman
4. La distància és la norma de (3)
5. Comparar la distància amb el llindar màxim
    - Si la distància està per sota del llindar màxim:
        - Si la distància està per sobre d'un llindar mínim i ja s'ha llegit prou posicions de càmera per calibrar el filtre Kalman, preneu la diferència entre (1) i (3) i useu-lo per actualitzar la posició de la càmera i desplaçar el vector (el vector *shift*).
        - Utilitzar el punt del món real del *camera pose* per actualitzar el filtre de Kalman.
        - Actualitzar el centre de la trajectòria (tant el punt com el pes) i obtenir el valor temporal pel centre (*weighted center*) que es farà servir per obtenir els angles.
        - Obtenir els angles dels fotogrames
        - Utilitzar els angles per:
            * Analitzar si ja hem finalitzat la captura de les dades

        * Analitzar si hem d'afegir o no el fotograma actual a la trajectòria. És a dir, comprovar si està en el rang d'angles actual.
        * Donar informació o *feedback* a l'usuari
      - Afegir la càmera a la trajectòria
    - Si no, és a dir, en el cas que la distància està per sobre del llindar màxim
      - Aturar la captura i comprovar [3]:
        (a) Si s'ha capturat la imatge principal
        (b) Si s'ha realitzat la captura de +30 fotogrames
      - Si se satisfan les dues condicions, la captura es considera exitosa i, per tant, es poden generar els *inputs* per l'API de LogMeal.
      - En cas contrari, s'ha de descartar totes les dades de la captura.
6. Si l'angle està situat en el rang d'angles actual, s'ha d'emmagatzemar el fotograma i la seva informació en la nostra estructura MyTuple. Per exemple, l'angle actual és 90 i el rang actual és [89.6, 90.4]
7. Si ja s'han capturat tots els fotogrames:
    - Generar els arxius per l'API
8. Si no, tornar al pas (1)

3.8 Crida a l'API

Si un usuari vol utilitzar el SDK implementat ha de realitzar el següent:

1. En primer lloc, ha de fer la crida al mètode *callLogMealSDK()*:

   ```
   Object[] response = logMealSDK.callLogMealSDK(session, frame, this);
   ```

   Aquesta crida retorna variables booleanes per controlar els següents estats de l'execució:
    - L'execució del procés ha començat.
    - L'execució del procés encara no ha finalitzat.
    - L'execució del procés ha finalitzat.
    - Ha canviat la trajectòria.
    - Ha finalitzat l'execució amb èxit o no.

   A més retorna informació sobre l'angle i el rang d'angles actual.
2. Comprovar si ha finalitzat o no l'execució:
    - Si no ha finalitzat l'execució tornar al pas 1.
    - Si ha finalitzat l'execució, comprovar si ha estat exitosa o no.

[3] Aquestes condicions també es comproven en el cas que l'usuari pel motiu que sigui decideixi voluntàriament aturar la captura

   - Si ha estat exitosa, l'usuari pot obtenir les dades necessàries per fer la crida a l'API
   - Si no, ha de tornar a fer la captura.

3. D'altra banda, existeix un cas opcional per la casuística en què l'usuari durant la captura prem el botó de finalitzar el procés. Si l'usuari vol controlar aquest cas, ha de cridar al mètode:

   ```
   logMealSDK.userStopButton();
   ```

   que comprova si estan els requisits mínims per fer una crida a l'API que hem comentat durant tot aquest document o no.

3.9 Consideracions

ArCore de vegades té errors de *tracking* o de seguiment. S'ha detectat l'existència dels petits errors de seguiment de la càmera que es poden corregir amb el filtre kalman, però els errors de seguiment grans restabliran el seguiment de la càmera i ja no serà possible relacionar un fotograma abans de l'error amb un fotograma després de l'error. És per aquest motiu que s'ha decidit descartar les dades anteriors i torna a iniciar la captura de dades si es produeix un error de seguiment i la cobertura de l'angle no és suficient.

Finalment, cal comentar que, si no se satisfan un mínim de condicions de lluminositat favorables, ArCore pot donar errors durant l'execució del procés.

Capítol 4

Resultats

En aquesta secció es comentaran i analitzaran de manera crítica tots els resultats obtinguts tenint en compte els objectius plantejats per aquest projecte 1.4.

En primer lloc, cal destacar que l'objectiu principal d'aquest projecte és la creació d'un SDK per a la representació d'objectes en 3D per integrar i dotar a l'API de LogMeal les eines necessàries per a realitzar la crida a un dels algoritmes més importants que és el *food quantity estimation*. Aquest objectiu s'ha portat a terme satisfactòriament perquè s'ha aconseguit generar els arxius necessaris tot complint els requisits que ens vam plantejar a l'inici d'aquest projecte. A continuació, explicarem els resultats obtinguts des de diverses perspectives:

1. Resultats obtinguts per l'API:
    - S'ha generat l'arxiu JSON, la seqüència de vídeo i la imatge principal amb els requisits destijats amb èxit.
    - S'ha realitzat el testatge amb l'API de LogMeal:
        - Com es pot observar en la figura 4.1 la crida a l'*endpoint* de depth recognition [1] es realitza correctament i s'assoleix un *status code* de 200 que vol dir que els arxius que hem generat s'han pogut enviar correctament a l'API.

FIGURA 4.1: Imatge que mostra la resposta de la crida a l'endpoint de l'API en *POSTMAN* .

[1] https://staging.api.logmeal.es/v2/image/segmentation/complete/quantity

– A continuació ens posem l'objectiu de dur a terme la captura d'una poma. Com que hem integrat el SDK a l'API s'observa que aquesta ens retorna una dada molt important com a respota per saber si s'ha realitzat correctament la captura:


```
"foodFamily": [
       {
           "id": 6,
           "name": "fruit",
           "prob": 0.75830078125
       }
   ],
   "imageId": 1622573,
   "processed_image_size": {
       "height": 360,
       "width": 480
   },
   "segmentation_results": [
       {
           "center": {
               "x": 321,
               "y": 230
           },
           "contained_bbox": {
               "h": 198,
               "w": 199,
               "x": 222,
               "y": 131
           },...,
   "recognition_results": [
               {
                   "foodFamily": [
                       {
                           "id": 6,
                           "name": "fruit"
                       }
                   ],
                   "foodType": {
                       "id": 2,
                       "name": "ingredients"
                   },
                   "hasNutriScore": true,
                   "id": 1253,
                   "name": "apple",
                   "nutri_score": {
                       "nutri_score_category": "A",
                       "nutri_score_standardized": 77
                   }
]
```

La figura 4.2 mostra la imatge d'una poma que s'ha enviat a l'API per testejar. S'observa Com es pot observar en part del JSON

FIGURA 4.2: Imatge d'una poma que representa la imatge principal d'una captura que s'envia a l'API.

retornat per l'API detecta correctament que és una poma, a més, ens dona dades com el centre de la poma que es pot observar que coincideix. La imatge té la mateixa resolució que el vídeo. A més, si l'execució no s'ha pogut enviar correctament a l'API, com que el SDK està integrat a aquesta retorna el *status code*. En aquest cas, ha retornat *Response code: 200* [2]. Finalment, convé subratllar l'atribut *imageId* que és molt important per a la crida posterior a un altre *endpoint* que mesura la quantitat estimada del menjar.

2. Resultats dels algorismes:

   - La implementació de l'algorisme de Kalman corregeix els problemes de *tracking* interns d'Android amb èxit. És important destacar que els canvis de trajectòria que es detecten amb el filtre de Kalman es controlen amb uns *thresholds* que depenen del dispositiu.
   - L'algorisme que calcula l'angle actual en el qual es troba la càmera en el món real és bastant precís[3]. Convé subratllar que la dificultat d'aquest algoritme radica en obtenir un resultat precís de manera dinàmica (recordem que la captura es realitza en temps real).
   - L'algorisme per obtenir el centre també és precís i si l'usuari a l'hora de fer una captura apunta tota l'estona cap al mateix centre, l'algorisme es comporta de manera constant.
   - Cal destacar que l'API no permet arxius de mida superior a 1 MB, per tant, es recomana que a l'hora de realitzar la captura els moviments siguin els més lents possibles per tal de capturar imatges que siguin les més semblants entre elles. Perquè si fem moviments grans pot passar que les imatges capturades en les posicions i, i+1 siguin molt diferents i a l'hora de crear la seqüència de vídeo no es pugui comprimir de manera òptima pel nostre algoritme, com a conseqüència es pot obtenir un arxiu de més de 1 MB. Cal recalcar que és un cas provat extrem a l'hora de fer el testatge exhaustiu del SDK.

[2] El *status code* 200 indica que la crida a l'API s'ha realitzat amb èxit

[3] menys en les primeres iteracions del procés, tal com s'ha comentat anteriorment

- Realitzar un moviment molt brusc pot canviar la trajectòria i, per tant, finalitzar el procés de captura.
- Les implementacions dels algoritmes estan pensades per realitzar trajectòries començant sempre des d'un angle de 50 graus aproximadament, d'esquerra a dreta, o viceversa. Començar el procés de captura des d'un altre punt pot portar a errors en el càlcul dels angles.
- Finalment, cal comentar que el SDK està pensat per fer una captura per procés executat.

A continuació, mostrarem imatges de la interfície que s'ha usat per al desenvolupament d'aquest SDK. Cal destacar que no s'ha buscat l'estètica sinó que sigui el més funcional i òptim per fer testatges. Si la captura no es realitza de manera correcta:

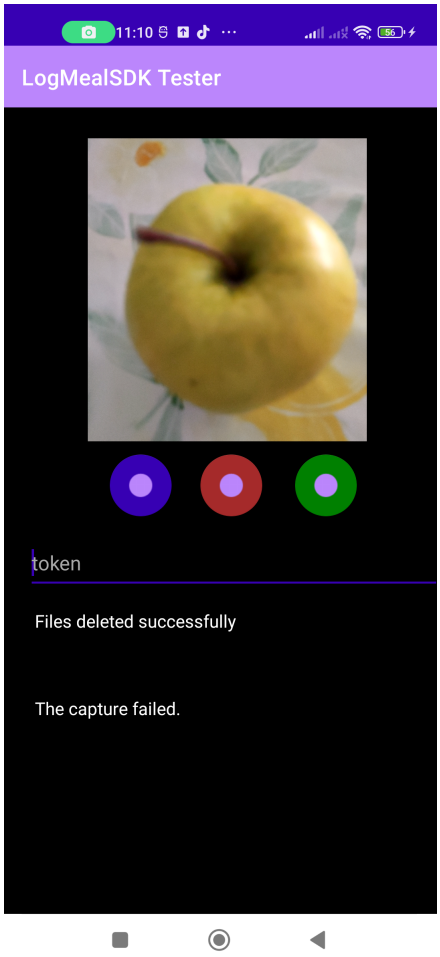

FIGURA 4.3: Imatge que mostra una captura no exitosa. En una captura que es considera no exitosa s'eliminen totes les dades prèviament recol·lectades.

La següent imatge mostra una interfície amb les dades de l'angle i rang d'angles actual. D'aquesta manera com a *developer* es pot visualitzar de manera molt senzilla si els algoritmes funcionen correctament o no:

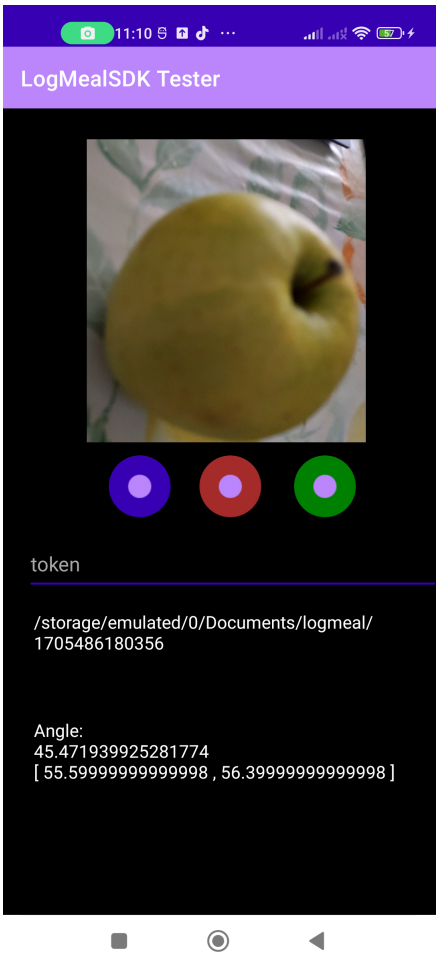

FIGURA 4.4: Imatge que mostra la interfície durant el procés de captura dinàmica.

Un cop es finalitza amb èxit la captura anterior s'ha d'esperar uns segons a què es processi la informació, sobretot la creació del vídeo. Recordem que estem fent una captura de manera dinàmica i no sabem mai l'estat futur de la captura, és per aquest motiu que la seqüència del vídeo es crea al final del procés. A continuació, per enviar la captura a l'API en el camp que posa *token* s'escriu un *token* proporcionat per LogMeal per fer la crida a l'API, i posteriorment se selecciona el botó verd.

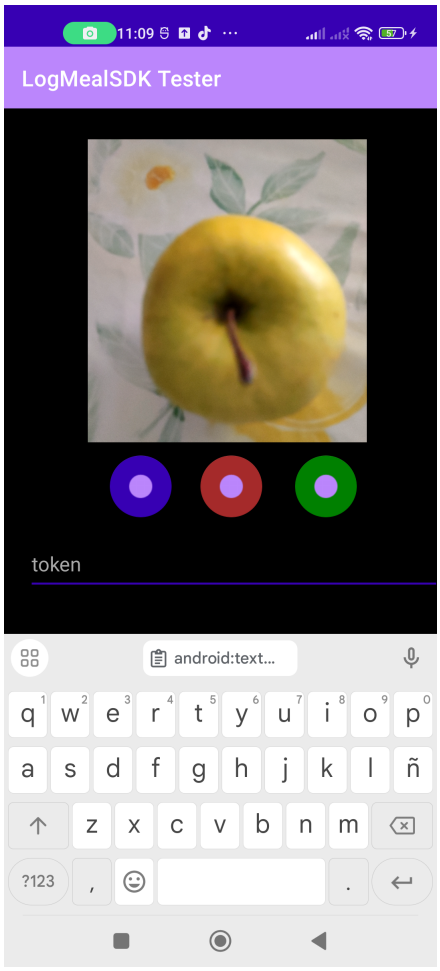

FIGURA 4.5: Imatge que mostra el camp de *token*.

Finalment, els resultats de l'API un cop s'ha fet el procediment anterior es poden visualitzar pel terminal de l'aplicació. Es pot visualitzar tant el *status code* com la informació que retorna l'API. Si és exitosa retorna una resposta, si no mostra un missatge amb l'error corresponent.

Capítol 5

Conclusions

En aquesta secció parlarem de les conclusions del projecte on comentarem si s'han satisfet els objectius inicials o no. A continuació, mostrarem els objectius que s'han realitzat amb èxit:

- S'ha realitzat el treball d'investigació sobre la llibreria ArCore: funcionament, avantatges, inconvenients...
- S'ha implementat un mètode per obtenir la informació essencial de la càmera.
- S'ha implementat d'un mètode per guardar, en memòria, un fotograma (partint de la càmera).
- S'ha realitzat en Android un model que permet fer captures de X fotogrames.
- S'ha realitzat la lògica per guardar els arxius d'una mateixa captura en una carpeta única.
- S'ha implementat la lògica per inicialitzar el centre d'una escena en una captura.
- S'ha implementat un mètode per actualitzar el centre de l'escena dinàmicament.
- S'ha implementat un mètode per aplicar un *weight* al centre de l'escena.
- S'ha implementat un mètode per inicialitzar *Kalman filter*.
- S'ha implementat un algoritme per calcular l'angle a partir dels atributs *altitude* i *azimuth* (versió inicial de l'algoritme).
- S'ha implementat un algoritme per calcular l'angle a partir del centre de l'escena i la primera càmera o fotograma (versió final de l'algoritme).
- S'ha implementat i investigat quin és el millor rang d'angles en termes d'usabilitat per a realitzar una captura de manera còmoda.
- S'ha implementat la seqüència de vídeo a partir d'una sèrie de fotogrames seleccionats.
- S'ha creat una estructura de dades per emmagatzemar tota la informació rellevant per a la creació del JSON final.
- S'ha implementat un mètode per seleccionar la *main image*.

- S'ha implementat la lògica de l'algoritme final on es posa en relació tots els algoritmes creats.
- S'ha integrat el SDK en l'API de LogMeal.

D'altra banda, cal comentar que en l'algorisme de filtre de Kalman, no s'ha aconseguit trobar un *threshold* general que sigui vàlid per tots els dispositius per tal de detectar un canvi de trajectòria, només és vàlid per dispositius amb paràmetres de la càmera semblants al dispositiu que s'ha usat pel desenvolupament. L'algorisme que calcula la distància en el filtre de Kalman depèn de la càmera del dispositiu, per tant, s'ha de fer un treball d'investigació per realitzar alguna mena de càlcul previ tenint en compte els paràmetres de la càmera i aconseguir trobar una relació amb el *threshold* amb l'objectiu de què sigui vàlid per tots els dispositius. Tot i que, no s'ha aconseguit trobar un *threshold* general, és important comentar que s'ha detectat aquest cas pel procediment de testatge del SDK que hem dut a terme.

Capítol 6

Treball futur

En aquest darrer capítol proposarem quina pot ser la continuació d'aquest projecte.

En primer lloc, es suggereix com a treball futur donar una solució al que s'ha comentat en l'últim paràgraf de la secció de resultats 5. És a dir, es recomana cercar una relació entre els llindars usats en el filtre de Kalman i entre els paràmetres de la càmera com poden ser el *principal point* i *focal length* entre altres. L'objectiu d'aquesta proposta és disposar d'uns llindars generals que es puguin ser funcionals per a qualsevol dispositiu.

D'altra banda, es proposa crear una interfície d'usuari en *React native* semblant a la imatge que es mostra en 2.8. Les línies que formen com una mena de circumferència que es mostren en aquesta imatge correspondrien als fotogrames que s'espera capturar en cada angle. Es suggereix adaptar aquesta circumferència a una trajectòria en forma d'arc. On cada línia verda indica la captura d'un fotograma de manera exitosa en el rang d'angles pertinent. A més, es mostraria les línies de color gris per indicar els fotogrames pendents per capturar en els rangs d'angles que encara no tenen cap fotograma.

Finalment, es mostraria en aquesta interfície tota la informació sobre l'estat de la captura seguint els següents requisits:

- S'ha de poder visualitzar quants fotogrames queden per tal de poder realitzar una captura exitosa.
- S'ha de poder detectar la velocitat de moviment de l'usuari.
- S'ha de poder informar a l'usuari si la captura no s'ha pogut completar de manera exitosa.
- S'ha de poder informar a l'usuari que un cop es finalitza la captura, si és exitosa, la crida a l'*endpoint* d'estimació del menjar de LogMeal tarda aproximadament uns 30 segons a donar una resposta perquè està processant les dades.

Bibliografia

Azarcoya-Cabiedes, Willy (2014). **?**Centre òptic**?** A: *ResearchGate*. URL: https://www.researchgate.net/figure/Pin-hole-camera-model-terminology-The-optical-center-pinhole-is-placed-at-the-origin_fig10_317498100.

Baeldung (2023). **?**Principal point**?** A: *baeldung*. URL: https://www.baeldung.com/cs/focal-length-intrinsic-camera-parameters#:~:text=the%20focal%20length%20is%20the,axis%20and%20the%20image%20plane.

Becker, Alex (2023). **?**Introduction to Kalman Filter**?** A: *Google*. URL: https://www.kalmanfilter.net/kalman1d.html.

Bhalaji Nagarajan Rupali Khatun, Marc Bolaños Eduardo Aguilar Leonardo Angelini Mira El Kamali Elena Mugellini Omar Abou Khaled Noemi Boqué Lucia Tarro Petia Radeva (jul. de 2021). **?**Nutritional Monitoring in Older People Prevention Services**?** A: *Springer Link*. URL: https://link.springer.com/chapter/10.1007/978-3-030-72663-8_5.

Google (2018). **?**ArCore**?** A: *Google*. URL: https://developers.google.com/ar/develop?hl=es-419.

La Vanguardia, Redactor de (2018). **?**Percentatge de malalties relacionades amb l'alimentació**?** A: *La Vanguardia*. URL: https://www.lavanguardia.com/vida/20180130/44409369489/la-oms-afirma-que-6-de-cada-10-enfermedades-tienen-relacion-con-alimentacion.html#:~:text=La%20OMS%20afirma%20que%206%20de%20cada%2010%20enfermedades%20tienen%20relaci%C3%B3n%20con%20alimentaci%C3%B3n.

Lee, Mike (2005). **?**Una buena salud empieza por una buena alimentación**?** A: *MyFitnessPal*. URL: https://www.myfitnesspal.com/es.

LogMeal (2017). **?**LogMeal Food AI**?** A: *LogMeal*. URL: https://logmeal.es/.

Ltd, JGraph (2000). **?**Draw io**?** A: *Draw io*. URL: https://app.diagrams.net/.

Moya, Pedro (2014). **?**Controla tu dieta con MyFitnessPal**?** A: *El Español*. URL: https://www.elespanol.com/elandroidelibre/aplicaciones/20140203/controla-dieta-milimetro-myfitnesspal-aplicacion-contar-calorias/250263_0.html.

Savino, Patricia (set. de 2011). **?**Obesitat i malalties no transmissibles relacionades amb la nutricio**?** A: *Scientific Electronic Library Online*. URL: http://www.scielo.org.co/scielo.php?pid=S2011-75822011000300005&script=sci_arttext.

Vuforia (2023). **?**Spatial Frame of Reference for Unity**?** A: *vuforia*. URL: https://developer.vuforia.com/library/device-tracking/spatial-frame-reference-native.

Wikipedia (2023). **?**Filtro de Kalman**?** A: *Wikipedia*. URL: https://es.wikipedia.org/wiki/Filtro_de_Kalman.

zeng, Max (2017). **?**Focal point**?** A: *gamedev.net*. URL: https://www.gamedev.net/forums/topic/692041-focal-point-question-computer-graphic/5355956/.