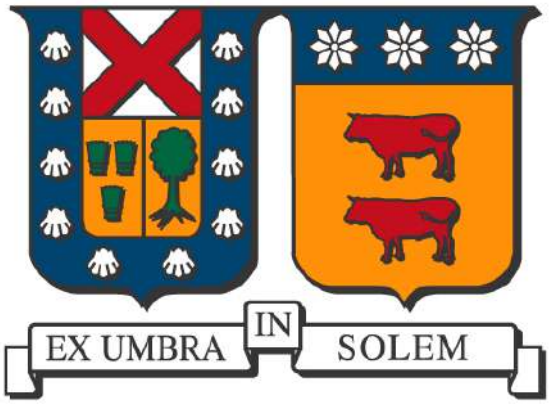
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
SANTIAGO - CHILE



"APLICACIÓN AIMES: AUTONOMÍA PARA USUARIOS CON DISCAPACIDAD VISUAL EN METRO DE SANTIAGO"

RODRIGO IGNACIO LOBOS CABELLOS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO/A CIVIL TELEMÁTICO/A

PROFESOR GUÍA:	**NICOLÁS TORRES RUDLOFF**
PROFESOR CORREFERENTE:	**NICOLÁS GÁLVEZ RAMIREZ**

AGOSTO - 2026

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☑ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Aplicación AIMES: Autonomía para usuarios con discapacidad visual en Metro de Santiago

Nombre del candidato(a): Rodrigo Ignacio Lobos Cabellos

Carrera / Grado: Ingeniería Civil Telemática

Campus: San Joaquín **Departamento:** departamento de electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Nicolás Torres Rudloff, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por **(marcar una opción):**

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

_____

_____

_____

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 27/10/2026 **Firma:** 

Estudiante o Candidato(a):

Fecha: 26-10-2026 **Firma:** 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Escaneado con CamScanner

Agradecimientos

Muchas personas me han ayudado a llegar hasta este punto. Mis amigos Rodrigo Contreras, Diego Flores, Benjamín Vallejos y Nilsson Acevedo, sin ellos me hubiera costado llegar a último año, o siquiera de haber terminado la carrera, fueron un pilar importante en momentos difíciles de la Universidad.

A mi familia, sobre todo a mi madre, fue la que estuvo apoyándome incondicionalmente en muchos sentidos, gracias a ella es que a día de hoy puedo tener la dicha de ser un futuro ingeniero.

A los que contribuyeron a sacar adelante este proyecto, el cual desde el inicio me pareció una idea la cual podía, y puede contribuir a la inclusión real, buscando la igualdad y equidad para todos

Finalmente, esta carrera y profesión se la dedico a mi padre, siempre quiso que sea mejor que él, pero a día de hoy pienso que nunca pude haber hecho todo lo que hizo para poder levantar una familia y darme la oportunidad de ingresar a telemática. Espero veas esto desde el cielo.

Resumen

Las personas con discapacidad visual enfrentan diversas dificultades para desplazarse de manera autónoma y segura en entornos urbanos complejos, donde la identificación oportuna de infraestructura y señaléticas resulta fundamental para una navegación sin inconvenientes. En el contexto del Metro de Santiago, elementos como ascensores, pisos podotáctiles, escaleras mecánicas y señaléticas direccionales constituyen referencias esenciales para la orientación y movilidad de estos usuarios. Sin embargo, su identificación depende de la asistencia de terceros o del uso de ayudas convencionales, las cuales presentan limitaciones para entregar información contextual en tiempo real.

Con el propósito de contribuir a esta problemática, el proyecto AIMES propone un sistema inteligente de asistencia para personas con discapacidad visual basado en técnicas de visión por computador e inteligencia artificial. La presente memoria aborda el desarrollo del componente de visión computacional encargado de detectar en tiempo real elementos relevantes del entorno, específicamente ascensores, pisos podotáctiles, escaleras mecánicas y señaléticas del Metro de Santiago correspondientes a indicaciones de dirección (más específicamente, a la dirección del andén y salida de la estación).

Para ello, se diseñó y construyó un conjunto de datos especializado mediante la recopilación, selección y etiquetado de imágenes representativas del entorno del Metro de Santiago. Posteriormente, se entrenaron y evaluaron modelos de detección de objetos basados en YOLOv11, comparando su desempeño mediante métricas de precisión y rendimiento computacional. Finalmente, el modelo seleccionado fue optimizado para su ejecución en dispositivos Android mediante TensorFlow Lite e integrado en una aplicación móvil desarrollada en Kotlin, capaz de realizar detección de objetos utilizando la cámara del dispositivo.

Como resultado, se obtuvo un prototipo funcional capaz de reconocer los principales elementos de accesibilidad considerados en este trabajo, alcanzando un valor de mAP@0.5 de 0,714 en la evaluación del modelo. En las pruebas de rendimiento, se alcanzó un máximo de 14,6 FPS en condiciones de referencia,

mientras que en las pruebas realizadas en terreno se obtuvieron velocidades entre 0,92 y 2,83 FPS, por debajo de los 20 FPS establecidos como requisito para una operación en tiempo real. Estos resultados evidencian la factibilidad de integrar modelos de visión por computador en dispositivos móviles para apoyar tareas de orientación y accesibilidad, aunque el rendimiento alcanzado y la ausencia de cuantización limitan actualmente su operación en tiempo real. La aplicación desarrollada constituye una base tecnológica para futuras mejoras orientadas a optimizar el rendimiento y avanzar hacia un sistema de asistencia para personas con discapacidad visual en el contexto del transporte público.

Abstract

People with visual impairments face various challenges when moving around independently and safely in complex urban environments, where the timely identification of infrastructure and signage is essential for smooth navigation. In the context of the Santiago Metro, elements such as elevators, tactile paving, escalators, and directional signage serve as essential reference points for these users' orientation and mobility. However, identifying these elements currently depends on assistance from others or the use of conventional assistive devices, which have limitations in providing real-time contextual information.

To address this issue, the AIMES project proposes an intelligent assistance system for people with visual impairments based on computer vision and artificial intelligence techniques. This thesis addresses the development of the computer vision component responsible for detecting relevant elements of the environment in real time—specifically elevators, tactile paving, escalators, and directional signage in the Santiago Metro indicating the direction to the platform and station exit.

To this end, a specialized dataset was designed and constructed by collecting, selecting, and labeling images representative of the Santiago Metro environment. Subsequently, object detection models based on YOLOv11 were trained and evaluated, with their performance compared using accuracy metrics and computational efficiency. Finally, the selected model was optimized for execution on Android devices using TensorFlow Lite and integrated into a mobile app developed in Kotlin, capable of performing real-time object detection using the device's camera.

As a result, a system was developed capable of recognizing the main accessibility features considered in this study, performing real-time inferences on a mobile device, and providing a technological foundation for future assistive systems designed for people with visual impairments. The developed app demonstrates the feasibility of integrating modern computer vision models into everyday devices, thereby promoting the development of solutions aimed at improving autonomy

and accessibility in public transportation systems.

Glosario

accesibilidad conjunto de condiciones que permiten que una persona pueda utilizar un entorno, servicio o sistema de manera autónoma, segura y comprensible, independiente de sus capacidades físicas, sensoriales o cognitivas.

aplicación nativa aplicación desarrollada directamente para un sistema operativo específico, como Android, utilizando sus lenguajes, librerías y APIs propias.

asistencia auditiva mecanismo de apoyo basado en la entrega de información mediante audio, utilizado en AIMES para comunicar instrucciones, confirmaciones y resultados de orientación.

asistencia visual tecnología o sistema orientado a apoyar a personas con discapacidad visual mediante captura, procesamiento e interpretación de información del entorno.

bounding box caja delimitadora generada por un modelo de detección para indicar la ubicación aproximada de un objeto dentro de una imagen.

cuantización Técnica de optimización de modelos de inteligencia artificial que reduce la precisión numérica de los parámetros y operaciones del modelo, por ejemplo, de valores de 32 bits a 8 bits. Esto permite disminuir el tamaño del modelo y los recursos computacionales necesarios, facilitando su ejecución en dispositivos con recursos limitados..

dataset conjunto de datos utilizado para entrenar, validar o probar modelos de visión por computador. En AIMES corresponde principalmente a imágenes etiquetadas del Metro de Santiago.

detección de objetos tarea de visión por computador que permite identificar y localizar elementos dentro de una imagen o video mediante etiquetas y cajas delimitadoras.

edge computing enfoque de procesamiento en el cual los datos se analizan directamente en el dispositivo local, reduciendo dependencia de servidores externos o conectividad.

frame imagen individual obtenida desde una secuencia de video, utilizada como entrada para el procesamiento visual del sistema.

inferencia proceso mediante el cual un modelo entrenado procesa una entrada, como una imagen o frame, para generar predicciones o detecciones.

orientación inmediata genera indicaciones auditivas orientadas a apoyar el desplazamiento del usuario hacia el objetivo indicado..

pipeline visual flujo de procesamiento encargado de capturar imágenes, preprocesarlas, ejecutar inferencia y postprocesar detecciones para obtener información útil del entorno.

piso podotáctil superficie con relieve utilizada como guía física para personas con discapacidad visual en espacios públicos, estaciones y recorridos peatonales.

postprocesamiento etapa posterior a la inferencia en la que se filtran, ordenan o ajustan las salidas del modelo para obtener detecciones finales utilizables por la aplicación.

preprocesamiento etapa previa a la inferencia donde una imagen se transforma, redimensiona o normaliza para cumplir con el formato de entrada requerido por el modelo.

retroalimentación auditiva respuesta entregada mediante audio para comunicar al usuario información sobre el estado del sistema, objetos detectados, posición relativa o instrucciones de orientación.

segmentación técnica de visión por computador que permite identificar regiones específicas de una imagen, generando máscaras asociadas a objetos o clases detectadas.

señalética conjunto de señales visuales utilizadas para orientar, informar o advertir dentro de un espacio físico, como estaciones del Metro de Santiago.

Transfer Learning Técnica de aprendizaje automático que consiste en utilizar un modelo previamente entrenado sobre un conjunto de datos o tarea y adaptarlo a una nueva tarea o conjunto de datos. Permite reducir el tiempo de entrenamiento y la cantidad de datos necesarios para obtener un modelo eficaz..

umbral de confianza valor mínimo utilizado para aceptar o descartar una predicción generada por un modelo de detección.

Acrónimos

AIMES Asistencia Inteligente en Metro de Santiago.

API Application Programming Interface.

APK Android Package Kit.

ASR Automatic Speech Recognition.

CPU Central Processing Unit.

CV Computer Vision.

DETR DEtection TRansformer.

DNN Deep Neural Network.

ENDIDE Encuesta Nacional de Discapacidad y Dependencia.

FPS Frames Per Second.

GPU Graphics Processing Unit.

GUI Graphical User Interface.

IDE Integrated Development Environment.

INE Instituto Nacional de Estadísticas.

IoT Internet of Things.

IoU Intersection over Union.

mAP mean Average Precision.

ML Machine Learning.

MVP Minimum Viable Product.

NMS Non-Maximum Suppression.

OCR Optical Character Recognition.

OMS Organización Mundial de la Salud.

PC Personal Computer.

PR Precision-Recall.

RAM Random Access Memory.

SAM Segment Anything Model.

SSD Single Shot MultiBox Detector.

STT Speech-to-Text.

TFLite TensorFlow Lite.

TTS Text-to-Speech.

UI User Interface.

UX User Experience.

ViT Vision Transformer.

WCAG Web Content Accessibility Guidelines.

YOLO You Only Look Once.

Tabla de Contenido

Glosario vi

Acrónimos ix

1 Introducción 1

1.1 Situación Actual 1

1.2 Desafíos del Entorno 2

1.3 Objetivos 4

1.3.1 Objetivo General 4

1.3.2 Objetivos Específicos 4

1.4 Estado del Arte 5

1.4.1 Contextual 5

1.4.2 Tecnológico 6

2 Requisitos del Sistema 16

2.1 Levantamiento de requisitos con usuarios 16

2.1.1 Visita a la Biblioteca Central para Ciegos 16

2.1.2 Entrevista con Usuario en Terreno 17

2.2 Requisitos Funcionales 18

2.2.1 Captura de información visual 18

2.2.2 Detección de elementos del entorno 18

2.2.3 Reconocimiento de texto 18

2.2.4 Interpretación de comandos de voz 19

2.2.5 Generación de retroalimentación auditiva 19

2.2.6 Funcionamiento local 19

2.3 Requisitos no Funcionales 19

2.3.1 Compatibilidad 19

2.3.2 Portabilidad 20
2.3.3 Confiabilidad 20
2.3.4 Usabilidad 20
2.3.5 Rendimiento 20
2.4 Requisitos de Interfaces 21
2.4.1 Cámara del dispositivo móvil 21
2.4.2 Aplicación móvil 21
2.4.3 Auriculares o salidas de audio 21
2.5 Requisitos de Desarrollo 22
2.5.1 Hardware 22
2.5.2 Software 23
2.6 Entorno de Soporte y Desarrollo 28
2.7 Perfiles de Usuario 28
2.7.1 Joven con Discapacidad Visual 29
2.7.2 Adulto con Discapacidad Visual 29
2.7.3 Persona Mayor con Discapacidad Visual 30
2.7.4 Cuidador o Familiar 30
2.7.5 Asistente o Trabajador del Metro 31

3 Diseño de la Solución 32
3.1 Diagramas de Conceptos 32
3.2 Marco general de la solución 33
3.2.1 Visión general de la solución 33
3.2.2 Diagrama de Contexto 34
3.2.3 Diagrama de Arquitectura 35
3.3 Evolución del prototipo y decisión de migración 36
3.3.1 Prototipo en computador: validación del pipeline visual 36
3.3.2 Prototipo inicial con Kivy/Buildozer 37
3.3.3 Migración a Android nativo con Jetpack Compose 37

3.4 Diseño de Interacción del Sistema 38
3.4.1 Principios de Diseño 38
3.4.2 Flujo de Interacción 38
3.4.3 Modelo conceptual de navegación e intención 39
3.5 Diseño técnico del componente de visión 41
3.5.1 Diseño del dataset 41
3.5.2 Diseño del modelo de detección 44
3.5.3 Diseño de la aplicación móvil 45
3.5.4 Diseño de Interfaz 46
3.6 Síntesis del diseño y transición a la implementación 51

4 Implementación de AIMES **52**
4.1 Construcción de Dataset 53
4.1.1 Recolección inicial de imágenes 53
4.1.2 Organización y Data augmentation en Roboflow 57
4.1.3 Evolución de versiones del dataset 58
4.2 Entrenamiento de Modelo 61
4.2.1 Configuración de entrenamiento 61
4.3 Métricas de evaluación 62
4.4 Conversion Modelo para App Móvil 64
4.4.1 Exportación a TensorFlow Lite 64
4.4.2 Cuantización y optimización 65
4.4.3 Validación con benchmark 65
4.5 Desarrollo de Aplicación en Kotlin 67
4.5.1 Entorno de desarrollo de aplicación 67
4.5.2 Arquitectura de la aplicación 70
4.5.3 Captura y procesamiento de imagen 73
4.5.4 Interpretación y orientación de dirección 76
4.5.5 OCR y confirmación de ruta 77

4.5.6 Comandos de voz y feedback de AIMES 80
4.5.7 Interfaz Móvil 82

5 Resultados **85**
5.1 Dataset para el Modelo 85
5.1.1 Distribución de Etiquetas 85
5.2 Modelo de Detección 87
5.2.1 Evaluación de Métrica: Precisión y Recall 87
5.2.2 Evaluación de Métrica: mAP@0.5 88
5.3 Evaluación de Métrica: F1 90
5.4 Matriz de confusión 91
5.5 Valores de benchmark 92
5.6 Aplicación y Uso en Terreno 94
5.6.1 Prueba simulada 94
5.6.2 Pruebas en terreno 95

6 Conclusiones **97**
6.1 Conclusión de Trabajo 97
6.2 Aportes 99
6.3 Limitaciones 100

7 Trabajo Futuro **101**
7.1 Optimización del modelo de detección 101
7.2 Ampliación y mejora del dataset 101
7.3 Mejora del reconocimiento mediante OCR 102
7.4 Comunicación de instrucciones al usuario 102
7.5 Optimización del flujo de procesamiento en tiempo real 103

Referencias

Anexos

.1 Pauta de entrevista en terreno a Don Iván . . . . . .

Índice de figuras

1.1 Entorno de circulación en una estación del Metro de Santiago donde convergen distintos elementos de orientación y accesibilidad. 3
3.1 Diagrama de Funciones 33
3.2 Diagrama de contexto del componente de visión por computador. 35
3.3 Arquitectura del componente de visión por computador. 36
3.4 Modelo de navegación: vista inicial del sistema 39
3.5 Modelo de navegación: confirmación de comando 40
3.6 Modelo de navegación: interacción durante la navegación 42
3.7 Interfaz Gráfica AIMES 47
3.8 Interfaz Gráfica AIMES 2 47
3.9 Interfaz Gráfica AIMES 3 48
3.10 Interfaz Gráfica Botón de Alerta 49
3.11 Paleta de colores Interfaz Gráfica AIMES 3 49
3.12 Interfaz de Desarrollador AIMES 50
4.1 Señales de Metro (1). 55
4.2 Señales de Metro (2). 55
4.3 Infraestructura de Metro (1). 55
4.4 Infraestructura de Metro (2). 56
4.5 Estación de Metro 56
4.6 Arquitectura de código de la aplicación AIMES implementada en Android. 70
4.7 Diagrama de flujo de la aplicación AIMES. 72
4.8 Flujo de Procesamiento de Frame. 73
4.9 Proceso de OCR: Selección de objeto de interés; Reconocimiento de texto; Normalización de caractéres; Coincidencia con estación esperada 79

4.10 Pantalla Principal de AIMES 82
4.11 Pantalla de Emergencia AIMES 83
4.12 Pantalla de Desarrollador AIMES 85
5.1 Distribución de instancias etiquetadas para las clases presentes en el dataset de señales-detección. 86
5.2 Comparación de modelos: parámetros, arquitecturas y resultados de mAP obtenidos. 87
5.3 Precision Recall de los Modelo YOLO. 88
5.4 F1 Score de los Modelo YOLO. 90
5.5 Matriz de confusión de los Modelo YOLO. 91
5.6 Capturas de la pantalla de desarrollador durante las pruebas simulada y en terreno. 94

Índice de tablas

1 Comparación de aplicaciones de asistencia visual 7
2 Tendencias tecnológicas en asistencia visual y relación con AIMES . . . 15
3 Stack tecnológico utilizado en el sistema AIMES 15
4 Evolución de versiones del dataset (v16–v23) 60
5 Lista de tecnologías utilizadas en la implementación de AIMES. 68
6 Comparación de mAP@0.5 entre arquitecturas 89
7 Resultados del benchmark de eficiencia computacional 92

1. Introducción

1.1. Situación Actual

En Chile, de acuerdo a la Encuesta Nacional de Discapacidad y Dependencia (ENDIDE, 2022), organizada por el Observatorio Social del Ministerio de Desarrollo Social y Familia [1], se presentan los siguientes resultados respecto a personas que presentan alguna discapacidad visual:

- Se estima que aproximadamente 4,6 millones de adultos presentan algún grado de pérdida de visión; de ellos, cerca de 153 mil viven con ceguera total.
- La Región Metropolitana concentra el 22,7 % de los adultos con pérdida de visión y el 24,9 % de los adultos con ceguera en Chile.
- 7,8 % de las personas ciegas usa bastón, y menos del 2 % utiliza tecnología accesible como lectores de pantalla.

Esta información contrasta con investigaciones hechas por la Organización Mundial de la Salud (OMS, 2023)[2]:

- En el mundo hay al menos 2.200 millones de personas con deterioro de la visión cercana o lejana.

El Metro de Santiago constituye uno de los principales medios de transporte público de la capital. Actualmente, su red está compuesta por siete líneas, 143 estaciones y aproximadamente 149 km de extensión, registrando un promedio de 2.146.206 pasajeros en un día laboral, según el último informe de Gestión del Directorio de Transporte Público Metropolitano (DTPM) [3].

En los últimos años, Metro ha incorporado diversas medidas orientadas a mejorar la accesibilidad, entre ellas ascensores, escaleras mecánicas y sistemas Braille. Además,

la mayoría de las estaciones dispone de ascensores, mientras que las estaciones de las líneas 3 y 6 cuentan con este tipo de infraestructura en su totalidad, destacando la incorporación de Braille para cargas bip! [4].

Sin embargo, la existencia de infraestructura accesible no resuelve por sí sola todas las dificultades que puede enfrentar una persona ciega o con discapacidad visual dentro de una estación. Elementos como la señalética direccional, la identificación de estaciones, la ubicación de ascensores y escaleras o la lectura de información textual requieren, en muchos casos, interpretar información visual.

Esta situación impone desafíos urgentes en materia de accesibilidad e inclusión, especialmente en espacios públicos de alta afluencia como infraestructura de transporte público el Metro de Santiago. En este escenario, se hace necesario potenciar nuevas iniciativas que permitan mejorar la calidad de vida y la autonomía de las personas con discapacidad visual, especialmente en su movilidad y acceso a servicios esenciales.

1.2. Desafíos del Entorno

Metro de Santiago constituye uno de los principales sistemas de transporte público del país y dispone de una infraestructura compuesta por ascensores, escaleras mecánicas, pisos podotáctiles y señaléticas direccionales que facilitan la orientación de los usuarios. No obstante, la identificación de estos elementos representa un desafío para las personas con discapacidad visual, especialmente en estaciones con alto flujo de pasajeros y múltiples alternativas de desplazamiento. En la figura 1.1 se aprecian distintas problemáticas al momento de movilizarse por Metro:

La principal problemática corresponde a la dificultad que presentan las personas con discapacidad visual para identificar de manera autónoma elementos de orientación y accesibilidad dentro del Metro de Santiago, lo que incrementa su dependencia de terceros y reduce la seguridad durante sus desplazamientos.

Factores de conflicto



Figura 1.1: *Entorno de circulación en una estación del Metro de Santiago donde convergen distintos elementos de orientación y accesibilidad.*

- **Limitaciones de señalética tradicional**: La señalética visual no es interpretada por personas ciegas o con baja visión, lo que les dificulta identificar salidas, destinos y obstáculos.
- **Entorno dinámico e impredecible**: el constante flujo de pasajeros y la existencia de múltiples rutas posibles dificultan la orientación.
- **Soluciones en infraestructura limitadas**: Recursos como los pisos podotáctiles y la señalización en braille constituyen ayudas importantes para la movilidad, pero no proporcionan información suficiente sobre todos los elementos relevantes presentes en una estación, como la ubicación de escaleras mecánicas, ascensores o señaléticas direccionales. [5]
- **Aumento del riesgo**: La imposibilidad de identificar obstáculos o rutas incrementa la probabilidad de accidentes y dependencia de terceros, lo que limita la autonomía.

En consecuencia, surge la necesidad de desarrollar herramientas tecnológicas que permitan identificar automáticamente estos elementos del entorno y entregar información accesible en tiempo real. En esta memoria se aborda este desafío mediante el desarrollo de un sistema de visión por computador capaz de detectar ascensores, pisos podotáctiles, escaleras mecánicas y señaléticas direccionales del Metro de Santiago

sobre dispositivos móviles.

1.3. Objetivos

1.3.1. Objetivo General

Desarrollar e implementar un sistema de visión computacional integrado en una aplicación móvil Android para la detección en tiempo real de elementos de orientación y accesibilidad del Metro de Santiago, mediante modelos de detección de objetos basados en Computer Vision, con el propósito de contribuir a la asistencia autónoma de personas con discapacidad visual.

1.3.2. Objetivos Específicos

- Construcción y evaluación del dataset
  Diseñar, construir y evaluar un dataset especializado para la detección de elementos relevantes del entorno del Metro de Santiago.

- Entrenamiento y evaluación de modelos de detección
  Entrenar y comparar modelos de detección de objetos basados en YOLOv11 en su versión nano (para su uso en aplicaciones móviles), evaluando su desempeño mediante métricas como precisión, recall, mAP y tiempo de inferencia, con el fin de seleccionar el modelo más adecuado para su implementación en dispositivos móviles.

- Optimización para dispositivos móviles
  Adaptar y optimizar el modelo de detección seleccionado para su ejecución local en dispositivos Android mediante TensorFlow Lite, considerando restricciones de rendimiento computacional y tiempo de respuesta.

- Desarrollo de la aplicación móvil
  Implementar una aplicación Android utilizando Kotlin que permita integrar el

modelo de detección entrenado, capturar imágenes mediante la cámara del dispositivo y realizar inferencias en tiempo real sobre el entorno del usuario.

- Validación de la Solución

  Evaluar el funcionamiento del sistema integrado mediante pruebas experimentales en escenarios representativos del entorno del Metro de Santiago, analizando la capacidad de detección de los elementos definidos y el desempeño de la aplicación móvil.

1.4. Estado del Arte

Esta sección considera dos aspectos importantes durante el desarrollo de AIMES: Contextual y Tecnológico. Ambos tienen el enfoque de establecer las bases con las que se desarrollan las funcionalidades, diseño de interfaz y definición del producto [6], y de integración de tecnologías en el sistema.

1.4.1. Contextual

Si bien el desarrollo de soluciones tecnológicas representa una oportunidad para mejorar la autonomía de las personas con discapacidad visual, su diseño debe considerar las características y necesidades de los usuarios finales. De acuerdo con la ENDIDE 2022, el uso de herramientas tecnológicas de accesibilidad, como lectores de pantalla o software especializado, es reducido dentro de esta población, lo que evidencia la importancia de desarrollar aplicaciones con interfaces simples, accesibles y orientadas a facilitar su utilización. [7]

Por otra parte, existe una serie de normativas regulatorias para respetar la accesibilidad a los servicios de transporte.

La Ley N°20.422 establece como uno de sus principios fundamentales la accesibilidad y el diseño universal de entornos, bienes y servicios. Asimismo, establece que

los edificios públicos y aquellos que atienden público deben ser accesibles y utilizables de manera autónoma por las personas con discapacidad [8]. En materia de transporte público, la legislación establece que los medios de transporte deben disponer de señalización, asientos y espacios suficientes y de fácil acceso, cuyas características deben ser determinadas mediante la reglamentación correspondiente.

Metro de Santiago ha desarrollado progresivamente distintas medidas destinadas a mejorar la accesibilidad de su red. Entre ellas se encuentran ascensores, rutas para personas ciegas, sistemas Braille y mecanismos de información sonora. De acuerdo con antecedentes publicados por el Servicio Nacional de la Discapacidad (SENADIS), Metro incorporó desde la década de 1990 medidas como ascensores, salvaescaleras, rutas para personas ciegas, sistemas Braille en pasamanos y torniquetes y sonorización de ascensores [9].

En 2018, Metro informó un plan de accesibilidad que contemplaba la instalación de 114 ascensores en 37 estaciones correspondientes a las líneas 1, 2 y 5. De acuerdo con SENADIS, los ascensores incorporados dentro de este plan contemplaban, además, intercomunicadores para situaciones de emergencia, cámaras de seguridad, información audible y Braille [10].

1.4.2. Tecnológico

El análisis tecnológico tiene como propósito identificar soluciones existentes y tecnologías aplicables al desarrollo de sistemas de asistencia para personas con discapacidad visual, con especial énfasis en aquellas relacionadas con percepción del entorno, reconocimiento de texto, interacción por voz y ejecución local en dispositivos móviles.

Primeramente, se considera hacer una revisión y comparativa entre las tecnologías del mercado actual las cuales tienen objetivos similares a AIMES.

La Tabla 1 muestra una comparativa de distintas soluciones tecnológicas respecto a AIMES, considerando distintos atributos como: Tipo de entrada y salida; Función en tiempo real, Navegación y uso offline. En el contexto local, destaca Lazarillo [11], [12],

una aplicación de navegación GPS accesible que opera actualmente dentro de la red de Metro de Santiago.

Aplicación	Entrada	Salida	Tiempo real	Navegación	Offline
Seeing AI [13]	Imagen	Voz	No (foto)	No	No
Be My Eyes [14]	Video en vivo	Voluntario remoto	Sí	No	No
Aira [15]	Video en vivo	Agente humano	Sí (humano)	Sí (humano)	No
Lookout (Google) [16]	Imagen + Video	Voz	Sí	No	Sí (parcial)
AIMES	Imagen + Video	Voz	Sí	Sí (Metro)	Sí

Tabla 1: *Comparación de aplicaciones de asistencia visual*

Con el propósito de fundamentar las decisiones tecnológicas adoptadas en esta memoria, se revisó el estado del arte de sistemas de asistencia para personas con discapacidad visual [17], [18], modelos de detección de objetos en tiempo real [19], herramientas para la construcción de conjuntos de datos y tecnologías orientadas al despliegue de modelos de inteligencia artificial en dispositivos móviles. Esta revisión permite justificar la selección de YOLOv11 para la detección de objetos, Roboflow para la gestión del conjunto de datos, TensorFlow Lite para la inferencia en Android y Kotlin como lenguaje para el desarrollo de la aplicación móvil. Las lecturas se dividen en dos secciones: Tecnologías y Software, que se enfoca en las herramientas y programas a utilizar en AIMES; e Implementaciones y trabajos relacionados, donde se revisan distintas implementaciones para obtener una visión general.

Tecnologías y Software

Lanzamiento de YOLOv11 (2024–2025) [20], [21]: La versión de YOLOv11 han incorporado mecanismos de atención más refinados, fusión de características jerárquicas y mejoras en el entrenamiento semi-supervisado. Estas versiones apuntan a una mayor robustez frente a condiciones adversas y menor dependencia de grandes conjuntos de datos anotados, fortaleciendo su aplicabilidad en soluciones en tiempo real sobre plataformas embebidas.

RepViT, integración de transformer (2024): RepViT (*Revisiting Mobile CNN From ViT Perspective*) es una familia de redes neuronales convolucionales ligeras diseñada para aplicaciones de visión por computador en dispositivos móviles y otros sistemas con recursos computacionales limitados [22]. La propuesta surge a partir de la revisión de las arquitecturas CNN móviles desde la perspectiva de los Vision Transformers (ViT), buscando incorporar algunas de sus decisiones de diseño sin abandonar las ventajas de las redes convolucionales. Esta arquitectura ofrece una buena relación entre precisión y eficiencia, lo que la convierte en una opción viable para reemplazar o complementar backbones convencionales en sistemas embebidos como AIMES, donde se requiere procesar simultáneamente detección, segmentación y OCR con bajo consumo de recursos.

Roboflow para construcción de datasets (2022): El trabajo *Roboflow 100: A Rich, Multi-Domain Object Detection Benchmark* [23] evidencia el uso de la plataforma Roboflow Universe como una fuente de datasets de visión por computador generados y anotados por la comunidad. Este conjunto de datos fue construido a partir de múltiples datasets públicos provenientes del ecosistema de Roboflow, abarcando diversos dominios de aplicación. El estudio destaca la importancia de contar con herramientas que permitan la creación y curación de datasets personalizados, debido a las limitaciones de los benchmarks tradicionales como COCO para representar escenarios reales. En este contexto, Roboflow se posiciona como una infraestructura que facilita la generación, gestión y diversificación de datasets para tareas de detección de objetos.

TensorFlow Lite (2019): TensorFlow Lite [24] ahora re-nombrado como LiteRT, es un framework de inferencia optimizado para ejecución de modelos de aprendizaje automático en dispositivos móviles y embebidos. Su diseño permite la ejecución eficiente de modelos mediante técnicas de optimización como cuantización y reducción de precisión, disminuyendo el tamaño del modelo y el consumo computacional. Este enfoque facilita la implementación de soluciones de visión por computador en tiempo

real directamente en el dispositivo, eliminando la dependencia de servidores externos y reduciendo la latencia. En este trabajo, TensorFlow Lite se utiliza para desplegar un modelo de detección basado en YOLOv11 en un entorno móvil, permitiendo su ejecución en condiciones de baja conectividad y recursos limitados.

Kotlin (JetBrains): Kotlin [25] es un lenguaje de programación moderno desarrollado por JetBrains y adoptado oficialmente por Google para el desarrollo de aplicaciones Android. Su diseño enfatiza la interoperabilidad con Java, la seguridad frente a valores nulos y una sintaxis concisa que reduce la complejidad del código. Estas características lo convierten en una opción adecuada para el desarrollo de aplicaciones móviles robustas y mantenibles, facilitando la implementación de sistemas en tiempo real como AIMES en dispositivos Android.

WCAG (2018): La *Web Content Accessibility Guidelines* [26] (WCAG) es un conjunto de directrices que promueve la perceptibilidad, operabilidad, comprensibilidad y robustez de los contenidos digitales. Este enfoque busca asegurar una experiencia inclusiva para personas con discapacidad visual, especialmente aquellas con baja visión o daltonismo, al garantizar que los sitios web y aplicaciones sean accesibles a través de diversas tecnologías de asistencia, como lectores de pantalla, así como mediante el uso de contrastes adecuados, navegación mediante teclado y otros métodos de interacción inclusivos.

CameraX (2019): CameraX [27] es una librería de Android desarrollada para aplicaciones que usen cámara, manteniendo compatibilidad con el ciclo de vida de la aplicación. Esta tecnología permite configurar una vista previa de cámara (**Preview**) y, al mismo tiempo, analizar frames en tiempo real, (**ImageAnalysis**), funcionalidad desarrollada específicamente para casos de uso como procesamiento de imágenes, visión artificial o inferencia de aprendizaje automático. Su uso es relevante en el desarrollo de la solución, ya que el sistema requiere capturar el entorno de manera continua, estable

y con baja latencia para alimentar el modelo de visión por computador.

ML Kit OCR (2025): Esta herramienta de Google [28] otorga reconocimiento de distintos tipos de texto, incluido el latín. Puede analizar el texto en distintas capas (bloque, línea, elemento o símbolo), identificar el idioma y hacer reconocimiento de texto en tiempo real. Este último resulta especialmente útil para el reconocimiento de texto en secuencias de video, permitiendo extraer y proporcionar la información textual asociada a un objeto una vez realizada su detección. De esta manera, la integración de esta herramienta permite complementar la información visual obtenida por el sistema y proporcionar datos relevantes sobre el contexto espacial del entorno.

TextToSpeech (TTS): API nativa de Android [29] para sintetizar texto en voz hablada o generar archivos de audio. Este se produce tras una inicialización asíncrona mediante un motor en segundo plano y su correcta gestión de ciclo de vida para la liberación de sus recursos. Esta tecnología permite abordar un requerimiento fundamental del sistema, correspondiente a la entrega de retroalimentación auditiva. Con este mecanismo, es posible proporcionar indicaciones, comunicar información relevante del entorno y generar respuestas en tiempo real.

SpeechRecognizer (STT): Igual que el TTS, esta clase tiene su origen en Android [30]. Si bien supone el proceso inverso, esta depende del hardware del dispositivo (micrófono), permisos de los tiempos de ejecución, y de motores locales de Google para poder procesar la información.

Jetpack Compose (2021): Es el kit de herramientas de Android [31] para construir UI nativas. El objetivo de este es simplificar y acelerar el desarrollo de UI, reduciendo la cantidad de códigos y sincronización de estados. Debido a la necesidad de implementar una interfaz dinámica, gestionar distintos estados de la aplicación y adaptar la interacción a las necesidades específicas del usuario, resulta necesario utilizar una

herramienta como Jetpack Compose para el desarrollo de la interfaz de AIMES.

Implementaciones y Trabajo Relacionado

Implementación de YOLO para ejecución remota (2023): Los autores de *Implementing Visual Assistant Using YOLO and SSD for Visually-Impaired Persons* [32] proponen una arquitectura cliente-servidor en la que los modelos de detección de objetos, como YOLO y SSD, se ejecutan en un servidor remoto, mientras que el dispositivo móvil actúa como cliente ligero. Las imágenes capturadas son enviadas al servidor para su inferencia, retornando posteriormente los resultados al dispositivo para su entrega mediante retroalimentación auditiva. Este enfoque permite reducir los requerimientos computacionales en el dispositivo móvil, aunque introduce dependencias de conectividad y latencia de red.

Uso de Transformers para Detección de Objetos Los autores de *Rep ViT: Revisiting Mobile CNN From ViT Perspective* presentan RepViT, una arquitectura orientada a la visión por computador en dispositivos móviles que reconsidera el diseño de las redes convolucionales móviles desde la perspectiva de los Vision Transformers (ViT) [22]. El trabajo parte de la necesidad de mejorar el equilibrio entre precisión y eficiencia computacional en modelos destinados a dispositivos con recursos limitados, donde aspectos como el costo computacional y el tiempo de inferencia son especialmente relevantes.

La propuesta analiza las características de las arquitecturas CNN móviles y estudia cómo determinados principios utilizados en los Vision Transformers pueden ser incorporados en su diseño. A partir de este análisis, los autores realizan modificaciones progresivas sobre la arquitectura, buscando obtener una estructura convolucional más eficiente sin depender de mecanismos de atención propios de los Vision Transformers. De esta manera, RepViT mantiene las ventajas asociadas a las redes convolucionales para escenarios móviles, mientras incorpora principios arquitectónicos derivados del

diseño de los ViT.

Desde la perspectiva de este trabajo, RepViT resulta relevante debido a su enfoque hacia arquitecturas compactas y eficientes para dispositivos móviles. La posibilidad de ejecutar modelos de visión directamente en el dispositivo permite reducir la dependencia de recursos externos y resulta especialmente pertinente en aplicaciones que requieren procesamiento en tiempo real.

Aplicación Móvil para la Accesibilidad a personas con discapacidad visual: Un trabajo especialmente relacionado con la propuesta de AIMES es el presentado por los autores de *Mobile App for Enhancing Accessibility Among the Visually Impaired* [33], quienes desarrollan una aplicación móvil orientada a personas con discapacidad visual mediante la integración de técnicas de visión por computador y procesamiento de texto. La propuesta incorpora un módulo de detección de objetos en tiempo real basado en YOLOv10 y un módulo de reconocimiento de texto mediante Tesseract OCR, cuyos resultados son posteriormente transformados en retroalimentación auditiva mediante un sistema de Text-to-Speech (TTS). Resulta relevante para AIMES debido a que demuestra la viabilidad de integrar diferentes técnicas de inteligencia artificial dentro de una aplicación móvil orientada a la asistencia de personas con discapacidad visual. Sin embargo, su aplicación no está enfocada en orientación espacial, sino más bien en la identificación de objetos y textos asociados a estas detecciones.

Sistema de navegación para personas no videntes: En el paper *A Navigation System for Visual Impaired People Using Object Detection* [34] se utilizan técnicas de detección de objetos en 360 grados, formando una grilla polar alrededor del usuario y haciendo detecciones en tiempo real de los objetos cercanos. En particular separa el frame y según la cercanía del objeto al foco, se determina la distancia que se tiene, tomando 3 zonas como posible riesgo: 5 metros, 3 metros y 1 metro. Esta técnica de proximidad de objeto otorga una referencia para poder determinar la lejanía o distancia a la que se encuentran los objetos detectados para poder entregar la retroalimentación

al usuario.

Discapacidad visual y percepción espacial: Los autores de *Visual Impairment Spatial Awareness System for Indoor Navigation and Daily Activities* [35] proponen utilizar técnicas de visión por computador para identificar objetos presentes en el entorno y determinar su ubicación relativa dentro del campo visual. Para ello, el frame capturado por la cámara se divide en una cuadrícula de 3×3 regiones, y la posición de cada objeto detectado se determina a partir del centroide de su bounding box. De esta manera, los objetos pueden ser clasificados según su ubicación, por ejemplo, en las regiones izquierda, central o derecha del campo visual. Esta representación permite transformar la información obtenida mediante la detección de objetos en una descripción espacial comprensible para el usuario. El sistema desarrollado en este proyecto tiene vital relevancia con lo propuesto en AIMES, que es la función de determinar espacialmente en dónde se encuentra el objeto, sin embargo el uso que se va a dar es en espacios públicos, punto a considerar si verdaderamente se puede aplicar en el uso diario de Metro.

Detección de infraestructura en trenes subterráneos La detección automática de elementos presentes en sistemas de transporte subterráneo también ha sido abordada mediante técnicas de aprendizaje profundo. Los autores de *Classification of Metro Facilities with Deep Neural Networks* [36] propusieron un sistema orientado a la clasificación automática de instalaciones presentes en túneles de Metro mediante redes neuronales convolucionales. El trabajo emplea una arquitectura basada en Inception V3 para extraer características visuales y clasificar distintos tipos de instalaciones presentes en este tipo de infraestructura.

En el contexto de AIMES, este enfoque constituye un antecedente para el reconocimiento automático de elementos de infraestructura dentro de las estaciones de Metro. Sin embargo AIMES busca ampliar la percepción del entorno hacia elementos relevantes para la navegación de personas con discapacidad visual, tales como escaleras,

ascensores, señalética y pavimento podotáctil.

En conjunto, los trabajos y tecnologías revisados proporcionan fundamentos relevantes para el desarrollo de AIMES, ya que abordan distintos componentes necesarios para construir una aplicación de asistencia destinada a personas con discapacidad visual. Las propuestas basadas en detección de objetos, segmentación semántica, reconocimiento de texto y análisis espacial permiten comprender y caracterizar el entorno, mientras que tecnologías como TensorFlow Lite, CameraX, Kotlin, Jetpack Compose, SpeechRecognizer y TextToSpeech posibilitan trasladar estas capacidades a una aplicación móvil con procesamiento local e interacción en tiempo real. Asimismo, las implementaciones estudiadas demuestran distintas estrategias para transformar la información visual en indicaciones comprensibles para el usuario. En este contexto, estos antecedentes sirven como base para diseñar AIMES como una solución orientada a entornos públicos complejos, particularmente el Metro de Santiago, donde la combinación de percepción visual, localización espacial, reconocimiento de señalización e interacción por voz permite entregar asistencia contextual, oportuna y accesible durante el desplazamiento del usuario.

A diferencia de los trabajos revisados, AIMES tiene un aporte escencial en tres aspectos en particular:

- Detección de señalética de Metro: Se harán detecciones orientadas específicamente a las señales que se tienen en Metro de Santiago.
- OCR de dirección: AIMES no sólo hará el reconocimiento de caractéres, sino que también utilizará esta información para orientar al usuario a través de Metro.
- Uso offline: AIMES operará y ejecutará el modelo de detección de manera local, haciendo que la solución se mantenga operativa durante todo el recorrido del usuario, evitando a su vez cortes o retardos en la asistencia del usuario.

En las Tablas 2 y 3 se exponen resúmenes de las tecnologías destacables y de interés

para este proyecto.

Tendencia	Descripción
Modelo de CV	Redes convolucionales optimizadas para dispositivos móviles con baja latencia y alta precisión.
OCR en tiempo real	Reconocimiento de texto en ambientes dinámicos como estaciones o paneles de trenes.
Fusión multimodal	Combinación de visión, texto y voz para una experiencia más robusta.
Interacción por voz personalizada	Asistentes con prompts adaptados al contexto y preferencias del usuario.
Edge Computing	Procesamiento interno del dispositivo para evitar dependencias.
Mapeo semántico del entorno	Reconocimiento contextual de objetos y señales para entender el entorno.
Framework multiplataforma	Desarrollo de aplicaciones móviles con GUI en Python, adaptable a dispositivos Android.

Tabla 2: *Tendencias tecnológicas en asistencia visual y relación con AIMES*

Tecnología	Rol en el sistema AIMES
YOLOv11	Modelo de detección de objetos en tiempo real optimizado para ejecución eficiente en entornos embebidos, utilizado para identificar elementos del entorno urbano.
RepViT	Arquitectura híbrida CNN-Transformer utilizada como alternativa eficiente para mejorar la extracción de características espaciales con bajo costo computacional.
TensorFlow Lite	Framework de inferencia en dispositivos móviles que permite ejecutar modelos de deep learning optimizados directamente en Android mediante cuantización y reducción de precisión.
Kotlin	Lenguaje de desarrollo oficial para Android utilizado en la implementación de la aplicación móvil, permitiendo integración con APIs del sistema y ejecución en tiempo real.

Tabla 3: *Stack tecnológico utilizado en el sistema AIMES*

2. Requisitos del Sistema

Previo a la definición de los requisitos, se realizó un proceso de levantamiento de información con usuarios y organizaciones vinculadas a la comunidad con discapacidad visual, con el fin de fundamentar las decisiones de diseño en necesidades reales.

2.1. Levantamiento de requisitos con usuarios

2.1.1. Visita a la Biblioteca Central para Ciegos

Durante la mañana del lunes 16 de junio se realizó una visita a la Biblioteca Central para Ciegos, guiada por su coordinador general, con el objetivo de observar prácticas cotidianas de personas con discapacidad visual y recoger información que permitiera orientar el diseño de AIMES.

Se identificó que la interacción con computadores se realiza mediante el lector de pantalla NVDA sobre Windows, en una navegación completamente auditiva y basada en teclado (mediante dactilografía, comandos y atajos), descartando por completo el uso del mouse. En dispositivos móviles, en tanto, se emplean los sistemas de accesibilidad nativos (VoiceOver en iOS y TalkBack en Android), siendo el teléfono el dispositivo de mayor adopción, evidenciado por su uso frecuente para conectarse a las clases impartidas por la propia Biblioteca.

Se destacaron además tres elementos relevantes para el diseño del sistema. Primero, la existencia de un estándar editorial para libros accesibles que regula colores, tipografías y tamaño de ilustraciones para personas con baja visión, lo que reafirma la importancia de considerar pautas de accesibilidad ya establecidas. Segundo, la relevancia de los pisos podotáctiles como guía de desplazamiento dentro del Metro, señalando explícitamente que su referencia dentro de AIMES no resulta redundante sino complementaria. Tercero, la necesidad de contemplar rutas alternativas o de emergencia, dado que las personas ciegas requieren planificar sus desplazamientos con anticipación ante even-

tuales cortes o fallas del servicio. Finalmente, se reafirmó que el uso del bastón es una práctica habitual y central en la movilidad independiente, por lo que el sistema debe concebirse como un complemento a esta herramienta y no como un reemplazo.

2.1.2. Entrevista con Usuario en Terreno

Se realizó una entrevista en terreno en la Línea 6 del Metro de Santiago a un usuario con discapacidad visual que se desplaza de forma independiente por la red, con un enfoque exploratorio y empático orientado a comprender sus hábitos, dificultades y expectativas frente a una eventual solución tecnológica como AIMES. Para la entrevista se hizo un cuestionario [.1], enfocado en conocer los requerimientos de un usuario que utiliza activamente el transporte público.
El usuario señaló que, pese a su autonomía y al uso habitual del bastón como herramienta principal de orientación, enfrenta dificultades recurrentes al interactuar con pantallas táctiles públicas (en particular, las máquinas de recarga Bip), donde no tiene certeza sobre el punto exacto en que está tocando la pantalla. También mencionó como una necesidad relevante contar con información sobre servicios cercanos dentro de las estaciones, como cajeros automáticos o locales comerciales, cuya ubicación actualmente se le dificulta encontrar. Otro aspecto destacado fue la dificultad para determinar en qué sector del andén queda posicionado tras bajar del tren, lo que complica su orientación dentro de la estación. Asimismo, indicó que sería de utilidad que el sistema permitiera identificar salidas específicas del Metro según su destino.

A partir de esta instancia de levantamiento con usuarios y organizaciones especializadas se definieron los perfiles de usuario y los requisitos funcionales y no funcionales presentados a continuación.

2.2. Requisitos Funcionales

2.2.1. Captura de información visual

El sistema debe permitir el uso de una cámara, ya sea integrada en un dispositivo móvil o mediante una cámara externa compatible. Esta debe posibilitar la captura continua y en tiempo real de imágenes del entorno.

Se espera que la captura de imágenes sea estable y frecuente. El sistema debe ser capaz de operar considerando factores propios del desplazamiento del usuario por las estaciones del Metro de Santiago, tales como variaciones en las condiciones de iluminación, movimiento de la cámara y cambios constantes en el entorno.

2.2.2. Detección de elementos del entorno

El sistema debe identificar señales, elementos de orientación y otros objetos relevantes mediante modelos de visión por computador. AIMES deberá integrar modelos de Computer Vision especializados en la detección de elementos visuales presentes en el entorno del Metro de Santiago.

2.2.3. Reconocimiento de texto

El sistema debe detectar y reconocer texto presente en señales y otros elementos relevantes del entorno mediante técnicas de Reconocimiento Óptico de Caracteres (OCR). AIMES deberá utilizar esta información para identificar nombres de estaciones, direcciones, números de andenes y otras indicaciones textuales presentes en el entorno del Metro de Santiago.

El texto reconocido deberá utilizarse como información contextual para determinar la ubicación y orientación del usuario dentro de la estación.

2.2.4. Interpretación de comandos de voz

El sistema debe recibir y procesar instrucciones verbales proporcionadas por el usuario mediante un sistema de reconocimiento de voz. AIMES deberá ser capaz de interpretar comandos relacionados con la navegación y orientación dentro del Metro de Santiago, tales como indicar un destino, solicitar orientación o iniciar y finalizar la guía.

2.2.5. Generación de retroalimentación auditiva

El sistema debe transformar la información obtenida mediante los módulos de percepción en indicaciones auditivas claras y comprensibles para el usuario. AIMES debe integrar un mecanismo de generación de respuestas por voz que permita comunicar de manera sencilla la información relevante detectada en el entorno.

Las indicaciones auditivas deben entregar información relacionada con la ubicación, orientación y navegación del usuario.

2.2.6. Funcionamiento local

Los principales módulos de procesamiento deben ejecutarse localmente en el dispositivo móvil, permitiendo que AIMES realice las tareas esenciales sin depender constantemente de servicios externos. Esto permite reducir la necesidad de transmitir información hacia servidores remotos y favorecer un procesamiento más rápido de los datos obtenidos desde los distintos módulos del sistema.

2.3. Requisitos no Funcionales

2.3.1. Compatibilidad

Los diferentes módulos que componen la solución deben poder integrarse y comunicarse de manera eficiente dentro del entorno Android utilizado por AIMES. Esta inte-

gración debe permitir el intercambio de información entre los módulos de percepción, procesamiento e interacción, asegurando un funcionamiento coordinado del sistema.

2.3.2. Portabilidad

La solución debe ser compatible con dispositivos móviles que utilicen el sistema operativo Android y que cuenten con el hardware necesario para ejecutar los distintos módulos de AIMES. Esto incluye disponer de recursos suficientes de procesamiento, memoria y capacidades de captura y reproducción de audio e imágenes.

2.3.3. Confiabilidad

El sistema debe mantener una operación estable y continua durante las sesiones de uso, asegurando el correcto funcionamiento de los procesos de captura de imágenes, inferencia de los modelos y generación de retroalimentación para el usuario. AIMES debe gestionar estos procesos de manera coordinada, evitando interrupciones que puedan afectar la orientación y navegación del usuario.

2.3.4. Usabilidad

La interacción con el sistema debe realizarse mediante comandos de voz claros y concisos, permitiendo al usuario comunicarse con AIMES de manera sencilla y natural. El sistema debe ser capaz de reconocer e interpretar estas instrucciones para ejecutar las acciones correspondientes sin requerir una interacción manual constante con el dispositivo.

2.3.5. Rendimiento

El tiempo de respuesta debe ser adecuado para acompañar el desplazamiento del usuario, permitiendo que las detecciones y la información obtenida se actualicen de forma continua. De esta manera, el sistema puede proporcionar asistencia contextual y

generar indicaciones de manera oportuna durante la navegación por las estaciones del Metro de Santiago.

2.4. Requisitos de Interfaces

2.4.1. Cámara del dispositivo móvil

La cámara del dispositivo móvil es utilizada para capturar imágenes del entorno del usuario de manera continua. Estas imágenes se usan como entrada para los módulos de visión por computador, permitiendo detectar y analizar señales, objetos y otros elementos relevantes para la orientación y navegación dentro de las estaciones del Metro de Santiago.

2.4.2. Aplicación móvil

La aplicación móvil constituye la interfaz principal de interacción entre el usuario y AIMES, integrando los distintos módulos que componen la solución. A través de ella se gestiona la recepción de comandos de voz, el procesamiento de la información obtenida del entorno, la configuración del sistema y la generación de asistencia auditiva para el usuario.

2.4.3. Auriculares o salidas de audio

Los auriculares o las salidas de audio del dispositivo son utilizados para proporcionar al usuario la retroalimentación generada por AIMES. A través de estos medios se entregan instrucciones de navegación, información contextual y avisos relevantes de manera clara y oportuna, permitiendo que el usuario reciba asistencia sin depender de elementos visuales.

2.5. Requisitos de Desarrollo

2.5.1. Hardware

Computador

Se necesitó un computador para realizar las principales actividades de desarrollo de AIMES, incluyendo la programación de la aplicación, preparación y entrenamiento de los modelos de visión por computador y reconocimiento de voz, así como las tareas de validación y pruebas del sistema. El equipo contó con recursos de procesamiento y memoria suficientes para ejecutar las herramientas de desarrollo y los procesos asociados al entrenamiento de los modelos.

Cámara

Se utilizó una cámara para realizar pruebas locales de los módulos de visión por computador, permitiendo simular la entrada de video que posteriormente fue obtenida desde la cámara del dispositivo móvil. Esta permitió evaluar el funcionamiento de los modelos de detección y procesamiento de imágenes bajo diferentes condiciones del entorno.

Auriculares o parlantes externos

Se utilizaron auriculares o parlantes externos para verificar el funcionamiento del sistema de retroalimentación auditiva. Estos dispositivos permitieron comprobar la generación y reproducción de las instrucciones mediante tecnología de conversión de texto a voz (Text-to-Speech), evaluando aspectos como claridad, volumen y oportunidad de las indicaciones entregadas.

2.5.2. Software

Herramientas para entrenamiento y preparación de modelos

Se utilizaron herramientas como OpenCV y Roboflow durante las etapas de procesamiento y preparación de imágenes, anotación de datos y construcción de los conjuntos utilizados para el desarrollo de los modelos de visión por computador. Estas herramientas permitieron preparar la información necesaria para el entrenamiento y la validación de los modelos empleados por AIMES.

Python

Python (versión 3.13.5) fue utilizado principalmente durante las etapas de desarrollo, entrenamiento, validación y conversión de los modelos de visión por computador. Además, permitió ejecutar pruebas y experimentos relacionados con el procesamiento de imágenes, la evaluación del rendimiento de los modelos y la preparación de los archivos utilizados posteriormente en la aplicación móvil.

OpenCV

OpenCV (versión 4.13.0.92) fue utilizado para el procesamiento y manipulación de imágenes durante las etapas de prototipado, preparación y prueba del componente de visión por computador. Su utilización permitió realizar operaciones sobre las imágenes de entrada y apoyar la evaluación de los modelos antes de su integración en el dispositivo móvil.

PyTorch

PyTorch (versión 2.13) fue utilizado como framework base para el entrenamiento y la evaluación de los modelos de detección implementados mediante Ultralytics. Su utilización permitió ejecutar los procesos asociados al aprendizaje de los modelos, gestionar los pesos entrenados y realizar las evaluaciones previas a su exportación hacia

formatos compatibles con la aplicación móvil.

Ultralytics

Ultralytics (versión 8.4.117) fue utilizado para el entrenamiento, validación, evaluación y exportación de las distintas configuraciones basadas en YOLOv11n. Este framework permitió gestionar el proceso de entrenamiento a partir de pesos preentrenados, obtener las métricas de desempeño de cada arquitectura y generar los modelos requeridos para las etapas posteriores de conversión.

ONNX

ONNX (versión 1.20.1) fue utilizado como formato intermedio durante el proceso de exportación de los modelos entrenados. Los modelos generados mediante Ultralytics fueron exportados inicialmente a ONNX y posteriormente convertidos al formato SavedModel de TensorFlow, desde el cual se generaron los archivos TensorFlow Lite utilizados en la aplicación móvil.

TensorFlow

TensorFlow (versión 2.21) fue utilizado durante el proceso de conversión de los modelos previamente exportados a ONNX. Mediante este framework se generaron representaciones compatibles con las herramientas de conversión de TensorFlow Lite, permitiendo obtener finalmente los modelos en formato `.tflite` requeridos por la aplicación Android.

psutil

psutil (versión 5.9.5) fue utilizado durante las pruebas de rendimiento de los modelos exportados. Esta librería permitió registrar el consumo de CPU y memoria RAM del proceso durante la ejecución del benchmark, complementando las mediciones de

latencia y cantidad de fotogramas procesados por segundo utilizadas para comparar las distintas arquitecturas.

Roboflow

Roboflow fue utilizado como plataforma web para la administración y preparación del conjunto de datos. Mediante esta herramienta se realizaron tareas de anotación, organización, versionado, preprocesamiento y aumento de datos, permitiendo mantener distintas versiones del dataset durante las etapas de entrenamiento y evaluación. Debido a su funcionamiento como servicio web, no se asoció una versión local específica a esta herramienta.

Android Studio

Android Studio [37] fue utilizado como entorno principal para el desarrollo de la aplicación móvil nativa de AIMES. Mediante este entorno se realizaron tareas de programación, compilación, depuración, ejecución de pruebas y generación de versiones destinadas al despliegue en dispositivos Android. El proyecto fue configurado mediante Android Gradle Plugin versión 8.13.0 y compilado utilizando Android SDK 36.

Kotlin

Kotlin (versión 2.0.21) fue utilizado como lenguaje principal para implementar la lógica de la aplicación móvil. Mediante este lenguaje se desarrollaron los componentes relacionados con la interacción por voz, el procesamiento de información, la integración de los módulos de visión por computador y el funcionamiento general de la aplicación.

Jetpack Compose

Jetpack Compose fue utilizado para la construcción de la interfaz de usuario nativa de AIMES. El proyecto utilizó Compose BOM versión 2024.09.00 para gestionar las

dependencias principales del framework, además de componentes configurados explícitamente en la versión 1.5.1 y Material3 versión 1.2.0. Su enfoque declarativo permitió implementar los componentes de la interfaz, gestionar los distintos estados de la aplicación y actualizar sus elementos visuales.

TensorFlow Lite (LiteRT)

TensorFlow Lite (versión 2.14.0), actualmente denominado LiteRT, fue utilizado para ejecutar localmente el modelo de detección en el dispositivo móvil. La aplicación incorporó además las bibliotecas TensorFlow Lite Support y TensorFlow Lite Task Vision en su versión 0.4.4. Esta integración permitió cargar y ejecutar los modelos en formato `.tflite` directamente en Android, reduciendo la dependencia de servicios externos durante el procesamiento visual.

CameraX

CameraX (versión 1.3.4) fue utilizado para gestionar la captura de imágenes provenientes de la cámara del dispositivo móvil. Mediante los componentes `Preview` e `ImageAnalysis`, esta librería permitió mantener la visualización de la cámara y obtener los fotogramas utilizados como entrada por el módulo de visión por computador.

ML Kit OCR

ML Kit Text Recognition (versión 16.0.0) fue utilizado para realizar el reconocimiento del texto contenido en la señalética detectada. La información obtenida permitió extraer nombres de estaciones y direcciones presentes en los letreros, complementando las detecciones del modelo y proporcionando información contextual para la validación de la dirección de viaje solicitada por el usuario.

SpeechRecognizer

SpeechRecognizer fue utilizado para implementar el reconocimiento de voz dentro de la aplicación. Al corresponder a una API nativa del framework de Android, no posee una versión independiente dentro del proyecto, sino que su disponibilidad se encuentra asociada a la versión del sistema operativo y del SDK utilizado. Esta API permitió capturar e interpretar los comandos verbales proporcionados por el usuario para interactuar con las principales funcionalidades de AIMES.

TextToSpeech

TextToSpeech fue utilizado para transformar la información generada por el sistema en retroalimentación auditiva. Al igual que SpeechRecognizer, corresponde a una API nativa del framework de Android y no a una dependencia con versionado independiente dentro del proyecto. Mediante esta API, AIMES pudo comunicar instrucciones de orientación, confirmaciones e información contextual a través de los dispositivos de salida de audio disponibles.

Android SDK

La aplicación fue desarrollada con Android SDK 36, utilizando `compileSdk = 36` y `targetSdk = 36`, mientras que la versión mínima soportada se estableció en `minSdk = 24`. Esta configuración permitió utilizar las APIs requeridas por los componentes de cámara, reconocimiento de texto, interacción por voz y ejecución local del modelo, manteniendo compatibilidad con dispositivos Android a partir del nivel de API 24.

Sistemas operativos

El desarrollo de AIMES se realizó principalmente sobre el sistema operativo Windows, donde se ejecutaron las herramientas de programación, entrenamiento y prepa-

ración de los modelos. Como plataforma objetivo de despliegue se utilizó Android, debido a que la aplicación móvil fue desarrollada como una solución nativa para dispositivos compatibles con este sistema operativo.

2.6. Entorno de Soporte y Desarrollo

El entorno de desarrollo de AIMES combinó hardware accesible, software de código abierto y herramientas de desarrollo móvil. El diseño, entrenamiento y las pruebas iniciales se realizaron en un computador de escritorio, mientras que los emuladores de Android permitieron validar distintas funcionalidades del sistema antes de su ejecución en dispositivos físicos.

Durante la etapa de prototipado se evaluó Kivy en conjunto con Buildozer para generar archivos APK. Sin embargo, esta alternativa fue descartada debido a limitaciones relacionadas con el acceso a APIs nativas y la integración de los módulos de audio y visión. La principal dificultad se presentó en la gestión de permisos de Android, ya que los conflictos producidos durante su solicitud provocaban el cierre inesperado de la aplicación y no pudieron ser resueltos durante esta etapa.

A partir de estas limitaciones, se optó por desarrollar la aplicación de forma nativa mediante Android Studio y Kotlin. Esta decisión facilitó la integración con las APIs de reconocimiento y síntesis de voz, CameraX, las librerías asociadas al procesamiento visual y los componentes propios del sistema operativo Android. Asimismo, permitió disponer de un mayor control sobre el ciclo de vida de la aplicación, los permisos y la integración entre los distintos módulos del sistema.

2.7. Perfiles de Usuario

Esta sección presenta los distintos perfiles de usuario identificados para el desarrollo de AIMES, considerando las características, necesidades y contextos de las personas con discapacidad visual que podrían utilizar la aplicación. Los rangos etarios fueron

determinados en base a la información obtenida del **Estudio de Personas con Discapacidad Visual en Chile según ENDIDE 2022** [7]. Estos antecedentes indican que la población con discapacidad visual se concentra principalmente en adultos y adultos mayores, aunque también se consideraron jóvenes adultos, quienes muestran mayor familiaridad con el uso de dispositivos móviles.

2.7.1. Joven con Discapacidad Visual

- **Socioeconómico y cultural:** Nivel socioeconómico diverso, pero prevalecen medio o medio bajo; generalmente con acceso a educación superior o media completa. [38]
- **Ocupacional:** Estudiante universitario o trabajador en formación.
- **Etario:** 18–29 años.
- **Características físicas, fisiológicas, psicológicas:** Se presume una buena salud física. Independiente en su vida diaria. Tiene autonomía en toma de decisiones. No posee afecciones mentales que afecten su independencia.
- **Otros:** Busca independencia total en el desplazamiento. Se infiere que utiliza smartphone.

2.7.2. Adulto con Discapacidad Visual

- **Socioeconómico y cultural:** Nivel socioeconómico diverso. Puede tener escolaridad media o técnica.[38]
- **Ocupacional:** Trabajador formal/informal o cesante.
- **Etario:** 30–59 años.
- **Características físicas, fisiológicas, psicológicas:** Capacidad física variable y sin afecciones mentales que puedan perjudicar su autonomía. Puede presentar

condiciones asociadas como glaucoma.

- **Otros:** Uso diario u ocasional de ayudas técnicas. Se presume que al menos está familiarizado con las nuevas tecnologías.

2.7.3. Persona Mayor con Discapacidad Visual

- **Socioeconómico y cultural:** Nivel socioeconómico bajo a medio. Poco familiarizado con la tecnología. Riesgo de exclusión digital.[38]
- **Ocupacional:** Jubilado o dependiente de apoyo familiar o estatal.
- **Etario:** 60 años o más.
- **Características físicas, fisiológicas, psicológicas:** Puede tener limitaciones físicas asociadas a la edad. Bajo manejo de tecnología. Posible comorbilidad (existencia de dos o más enfermedades).
- **Otros:** Requiere sistema simplificado. Mayor necesidad de soporte o asistencia con la tecnología.

2.7.4. Cuidador o Familiar

- **Socioeconómico y cultural:** Nivel socioeconómico diverso. Generalmente de entorno directo del usuario final. Compromiso afectivo.[38]
- **Ocupacional:** Familiar, acompañante o cuidador contratado.
- **Etario:** 20–65 años.
- **Características físicas, fisiológicas, psicológicas:** Buen estado físico y mental. Capacidad para asistir y supervisar.
- **Otros:** Rol secundario pero clave en configuración y soporte del sistema.

2.7.5. Asistente o Trabajador del Metro

- **Socioeconómico y cultural:** Empleado del sistema de transporte público. Capacitado en atención a usuarios.[38]
- **Ocupacional:** Personal de atención al pasajero.
- **Etario:** 25–60 años.
- **Características físicas, fisiológicas, psicológicas:** Buen estado físico. Formación en atención al público. Capacidad para resolver situaciones de emergencia.
- **Otros:** Puede actuar como punto de apoyo ocasional. No opera directamente el sistema.

Se han identificado dos categorías de usuarios: principales, que incluyen jóvenes, adultos y adultos mayores con discapacidad visual; y secundarios, conformados por cuidadores, familiares y asistentes del metro. Sobre la base de esta clasificación, se ha determinado que el enfoque principal será el adulto con discapacidad visual, dado que la prevalencia de esta condición es significativamente mayor en dicho grupo etario.[1], [7]

3. Diseño de la Solución

Este capítulo presenta el diseño de la solución desarrollada para AIMES, abordando desde su definición conceptual y funcional hasta la materialización de un sistema ejecutable en dispositivos Android.

Con el propósito de acotar el alcance del trabajo y construir un conjunto de datos representativo, se seleccionaron principalmente estaciones de las líneas 6 y 3, cuya infraestructura presenta criterios de diseño relativamente homogéneos, facilitando la recopilación y el etiquetado de imágenes para el entrenamiento de los modelos de detección.

El trabajo se enfocó en transformar un prototipo conceptual en un sistema capaz de capturar video en tiempo real, ejecutar inferencia local sobre los frames obtenidos y entregar orientación mediante retroalimentación auditiva clara y oportuna para el usuario. Para ello, se presenta inicialmente la definición de las funciones y conceptos que estructuran la solución, seguida de su arquitectura general, la descripción de los módulos que la componen y su relación con los requisitos funcionales establecidos.
Posteriormente, se describe la evolución del prototipo y las decisiones que condujeron a la migración hacia una aplicación Android nativa, junto con el diseño de interacción y los mecanismos de navegación e interpretación de intenciones. Finalmente, se detalla el diseño técnico del componente de visión por computador, incluyendo la construcción del dataset, el diseño del modelo de detección, la aplicación móvil y el flujo de integración de la información, para cerrar con la delimitación del alcance efectivamente implementado.

3.1. Diagramas de Conceptos

A continuación, se presenta el diagrama de conceptos desarrollado para el sistema AIMES, el cual permite explorar distintas soluciones técnicas posibles para cumplir con

las funciones clave del producto. Según Ulrich y Eppinger, un diagrama de conceptos "representa gráficamente las ideas o soluciones alternativas que pueden implementarse para lograr las funciones del producto", ayudando a visualizar combinaciones viables de tecnologías antes de definir un diseño final. En este caso, el diagrama permite organizar y analizar propuestas asociadas a accesibilidad, orientación espacial, interacción con el entorno y eficiencia operativa [39].

La Figura 3.1 muestra la situación ideal donde se denotan las funcionalidades y operación del sistema.

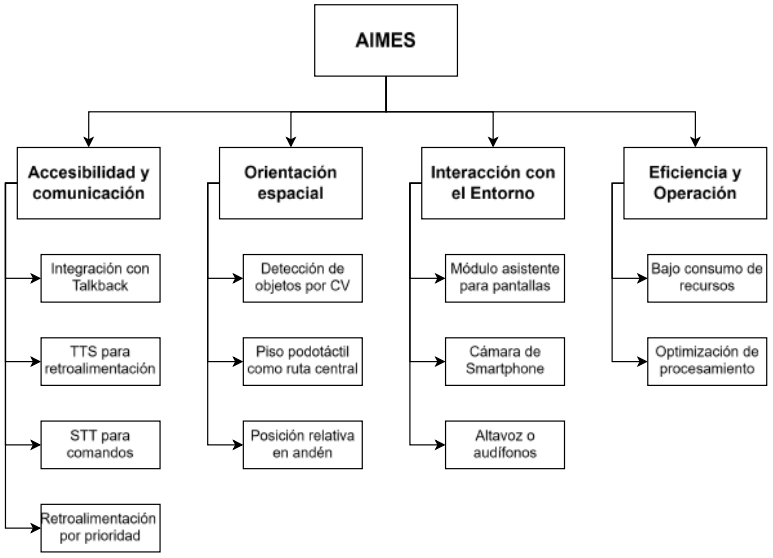


Figura 3.1: *Diagrama de Funciones*

3.2. Marco general de la solución

3.2.1. Visión general de la solución

AIMES es un sistema de asistencia orientado a apoyar el desplazamiento autónomo de personas con discapacidad visual dentro de la red de Metro de Santiago, mediante la combinación de Computer Vision, OCR y procesamiento de voz, ejecutados de forma local en un dispositivo móvil. A diferencia de soluciones que dependen de infraestruc-

tura externa o de conectividad permanente, AIMES fue concebido para operar como un asistente autocontenido: el propio teléfono del usuario captura el entorno a través de su cámara, interpreta los elementos relevantes para la orientación, y traduce esa información en indicaciones auditivas, sin requerir que el usuario interactúe visualmente ni de forma táctil con la pantalla del dispositivo.

La solución se estructura en torno a un ciclo continuo: el sistema detecta el entorno mediante un modelo entrenado específicamente para el contexto del Metro de Santiago, contrasta esa información con el destino solicitado por el usuario mediante comandos de voz, y genera instrucciones habladas que guían su desplazamiento hacia el objetivo indicado. Este enfoque responde directamente a las necesidades identificadas durante el levantamiento de información con usuarios y organizaciones especializadas, particularmente en lo relativo a mantener el bastón guía como herramienta principal de movilidad y a operar bajo los mismos principios de accesibilidad auditiva que ya utilizan los usuarios ciegos en sus dispositivos móviles, como TalkBack.

A partir de este punto, se revisa el diseño y estructuración de AIMES

3.2.2. Diagrama de Contexto

La Figura 3.2 presenta el contexto funcional del componente de visión por computador desarrollado en esta memoria. El sistema tiene como principal entrada el flujo de imágenes capturado mediante la cámara de un dispositivo móvil Android, el cual es procesado por un modelo de detección de objetos basado en YOLOv11. Como resultado, el sistema identifica en tiempo real elementos de orientación y accesibilidad presentes en el entorno del Metro de Santiago, tales como ascensores, escaleras mecánicas, pisos podotáctiles y señaléticas direccionales.

La salida del sistema corresponde a las detecciones realizadas por el modelo, las cuales son procesadas por la aplicación móvil para generar una representación visual de los objetos identificados y servir como base para futuras funcionalidades de asistencia dentro del proyecto AIMES. De esta manera, el componente desarrollado constituye

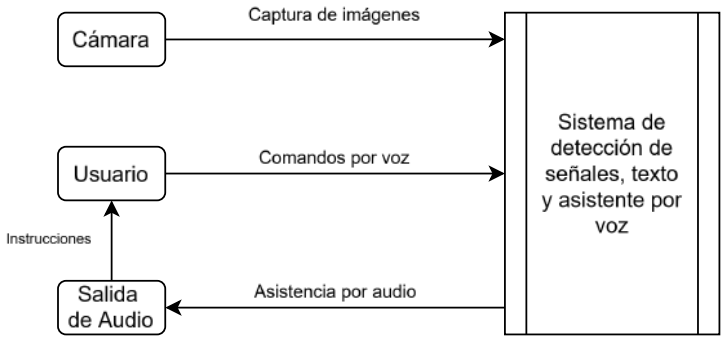


Figura 3.2: *Diagrama de contexto del componente de visión por computador.*

el módulo encargado de transformar la información visual capturada por la cámara en información estructurada sobre el entorno.

3.2.3. Diagrama de Arquitectura

La Figura 3.3 presenta la arquitectura del componente de visión por computador implementado en este trabajo. El flujo de imágenes obtenido desde la cámara del dispositivo móvil es procesado inicialmente por un módulo de adquisición de imágenes, encargado de capturar y preparar cada frame para su procesamiento. Posteriormente, las imágenes son enviadas al modelo de detección de objetos, previamente entrenado con un conjunto de datos construido específicamente para el entorno del Metro de Santiago.

El modelo realiza la inferencia sobre cada imagen para detectar los elementos de interés definidos durante la construcción del conjunto de datos. Las detecciones obtenidas son posteriormente interpretadas por la aplicación Android, desarrollada en Kotlin, la cual genera la representación gráfica de los objetos identificados y administra el flujo de inferencia en tiempo real.

Con el objetivo de permitir la ejecución del modelo directamente sobre el dispositivo móvil, el modelo entrenado fue convertido al formato .tflite (TensorFlow Lite), reduciendo los requerimientos computacionales y permitiendo realizar inferencias lo-

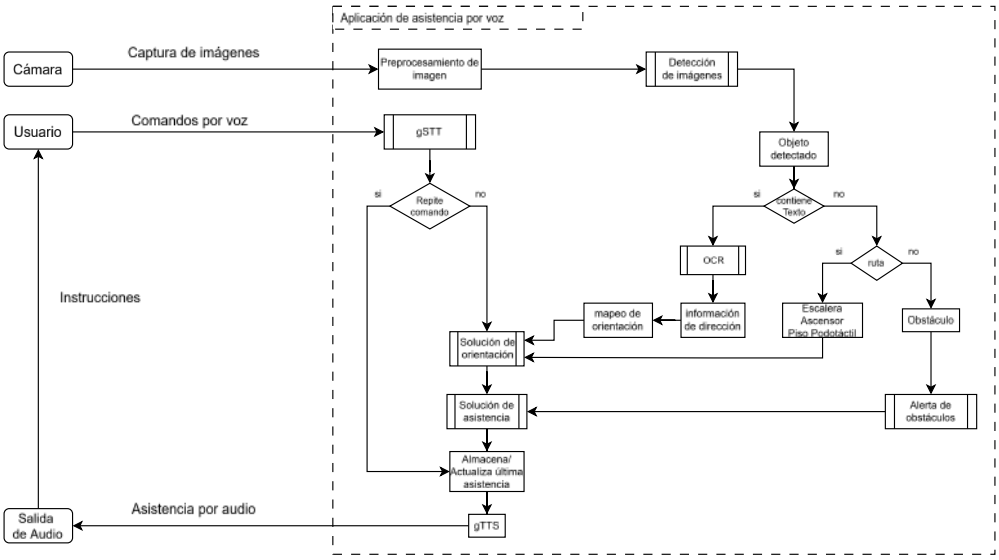


Figura 3.3: *Arquitectura del componente de visión por computador.*

cales sin depender de una conexión a internet.

3.3. Evolución del prototipo y decisión de migración

3.3.1. Prototipo en computador: validación del pipeline visual

Previo a cualquier desarrollo orientado a dispositivos móviles, se construyó un prototipo de escritorio en Python con el objetivo de validar el pipeline de percepción visual de forma aislada, sin las restricciones propias de un entorno móvil. Este prototipo permitió probar de manera temprana la detección de elementos del entorno mediante OpenCV y un primer modelo de visión por computador, así como evaluar la viabilidad de integrar reconocimiento de texto sobre imágenes capturadas desde una cámara externa. El objetivo de esta etapa no era obtener una aplicación utilizable, sino reducir el riesgo técnico del proyecto: confirmar que el enfoque de detección y orientación propuesto era viable antes de invertir esfuerzo en su portabilidad a un dispositivo móvil.

3.3.2. Prototipo inicial con Kivy/Buildozer

Una vez validado el pipeline visual, se evaluó Kivy [40] como framework para el desarrollo de una primera versión móvil de AIMES, empaquetada mediante Buildozer para su distribución como APK, apoyándose en herramientas como Plyer [41] para el acceso a APIs nativas del dispositivo, RHVoice [42] para la síntesis de voz y Vosk [43] para el reconocimiento de voz. La elección inicial de esta herramienta respondió a la posibilidad de reutilizar directamente el código Python desarrollado en la etapa anterior, evitando una reescritura temprana en otro lenguaje.

Sin embargo, durante el proceso de compilación y empaquetado, el prototipo presentó fallos recurrentes que impidieron obtener una versión funcional de la aplicación, incluso en sus funcionalidades más básicas. Tras estos intentos, se decide descartar Kivy como framework de desarrollo, antes de que se lograra obtener un prototipo móvil funcional bajo este enfoque.

3.3.3. Migración a Android nativo con Jetpack Compose

A partir de las limitaciones identificadas en la etapa anterior, se optó por migrar el desarrollo hacia Android nativo, utilizando Kotlin como lenguaje principal y Jetpack Compose para la construcción de la interfaz de usuario. Esta decisión permitió un acceso directo a los componentes de accesibilidad del sistema operativo, así como una integración nativa con las APIs de reconocimiento y síntesis de voz, la librería CameraX para la captura de video, y TensorFlow Lite para la ejecución local del modelo de detección de objetos previamente entrenado. La migración a un entorno nativo permitió, además, un mayor control sobre el rendimiento y la latencia del sistema, aspectos identificados como críticos para el correcto funcionamiento de la asistencia en tiempo real.

3.4. Diseño de Interacción del Sistema

3.4.1. Principios de Diseño

El desarrollo de la interfaz se guía por los siguientes principios:

- **Accesibilidad:** Toda funcionalidad debe estar disponible mediante entrada y salida por voz, sin requerir visión ni interacción táctil compleja.
- **Claridad y simplicidad:** Las respuestas deben ser breves y fácilmente comprensibles.
- **Retroalimentación continua:** El sistema debe entregar confirmaciones auditivas claras tras cada acción, hasta 2 repeticiones de confirmación son dadas por defecto; Si el usuario no confirma y entrega una nueva acción al sistema, la solución prosigue con esa nueva acción dada.
- **Personalización:** El usuario podrá configurar parámetros como volumen de voz, velocidad del habla, entre otros.

3.4.2. Flujo de Interacción

El diseño de interacción de AIMES se estructura en torno a tres momentos sucesivos: la activación del sistema, la confirmación de que el comando del usuario fue correctamente interpretado, y la ejecución de la navegación propiamente dicha. Cada uno de estos momentos se modela mediante un diagrama independiente, descritos a continuación, que en conjunto conforman el modelo conceptual de navegación e intención del sistema.

3.4.3. Modelo conceptual de navegación e intención

Vista inicial del sistema

El diseño de interfaz inicial (Fig. 3.4) define el estado de reposo del sistema, es decir, la condición en la que la aplicación permanece antes de que el usuario decida iniciar una sesión de asistencia. En este estado, el sistema no ejecuta ningún proceso de detección visual ni de navegación; su única función es mantenerse a la espera del comando de activación “Iniciar guía”. La decisión de diseño detrás de esta pantalla responde directamente a las características del público objetivo: al tratarse de un sistema orientado a personas con discapacidad visual, cualquier paso adicional de configuración manual o navegación por menús táctiles previos a la activación representaría una barrera de uso. Por ello, el punto de entrada se reduce a una única acción por voz, minimizando la carga cognitiva y motriz requerida para comenzar a utilizar la asistencia.

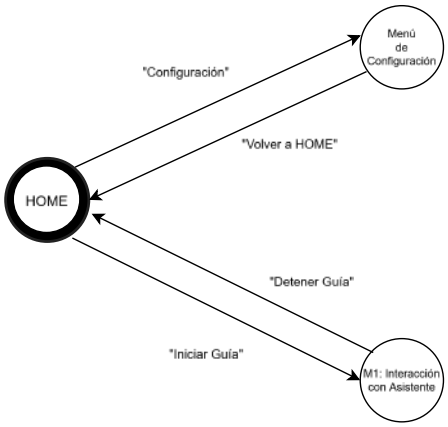


Figura 3.4: *Modelo de navegación: vista inicial del sistema*

Confirmación de comprensión del comando

Una vez activado el sistema, se introduce un segundo mecanismo de interacción (Fig. 3.5) orientado a verificar que la instrucción entregada por el usuario fue interpretada correctamente antes de continuar con cualquier acción de navegación. Tras recibir el comando de inicio, el sistema entrega una breve explicación de su funcionamiento

(salvo que el usuario opte por omitirla) y posteriormente transita a un estado de espera de nuevas instrucciones. Cada comando recibido en este punto es sometido a un proceso de confirmación explícita: solo si el sistema determina que comprendió correctamente la instrucción se avanza hacia el módulo de navegación; en caso contrario, el flujo retorna a un estado de espera o finaliza la interacción. Este mecanismo evita que un error de reconocimiento de voz no detectado se traduzca en una guía de navegación incorrecta, situación que en un contexto real podría derivar en una desorientación del usuario.

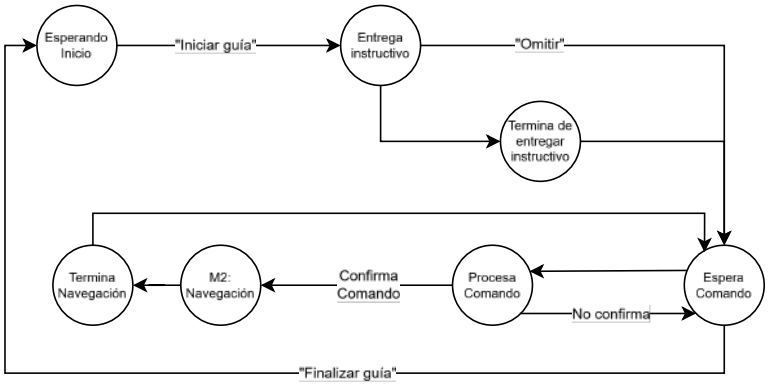


Figura 3.5: *Modelo de navegación: confirmación de comando*

Interacción durante la navegación

El tercer diseño (Fig. 3.6) modela el comportamiento del sistema una vez que la navegación ha sido confirmada e iniciada, y constituye el componente más complejo de los tres al integrar múltiples rutas de ejecución según el tipo de comando emitido por el usuario. Se identifican tres comandos principales, cada uno asociado a una rama de interacción distinta:

- El comando “Me dirijo a..., estoy en..”, inicia un ciclo de detección de señalética en el entorno, seguido de una etapa de verificación de la dirección indicada en dicha señalética; al reconocerse un letrero con la información “Dirección a..”, el sistema continúa la navegación hasta alcanzar el destino especificado.
- El comando “Quiero utilizar escaleras” replica una lógica equivalente, pero orientada a la detección de escaleras como elemento de referencia, seguida igualmente de una verificación de dirección antes de guiar al usuario a través de ellas.
- El comando “Quiero ir a la salida...” reutiliza el ciclo de detección de señalética, pero condicionado al reconocimiento específico de letreros del tipo “Salida a...”.

3.5. Diseño técnico del componente de visión

3.5.1. Diseño del dataset

El desarrollo del sistema requiere la construcción de un conjunto de datos especializado debido a la inexistencia de repositorios públicos orientados a la detección de elementos de orientación presentes en el Metro de Santiago. En consecuencia, se diseña un proceso de captura y organización de imágenes que permitiera representar la infraestructura relevante para el entrenamiento del modelo de detección de objetos.

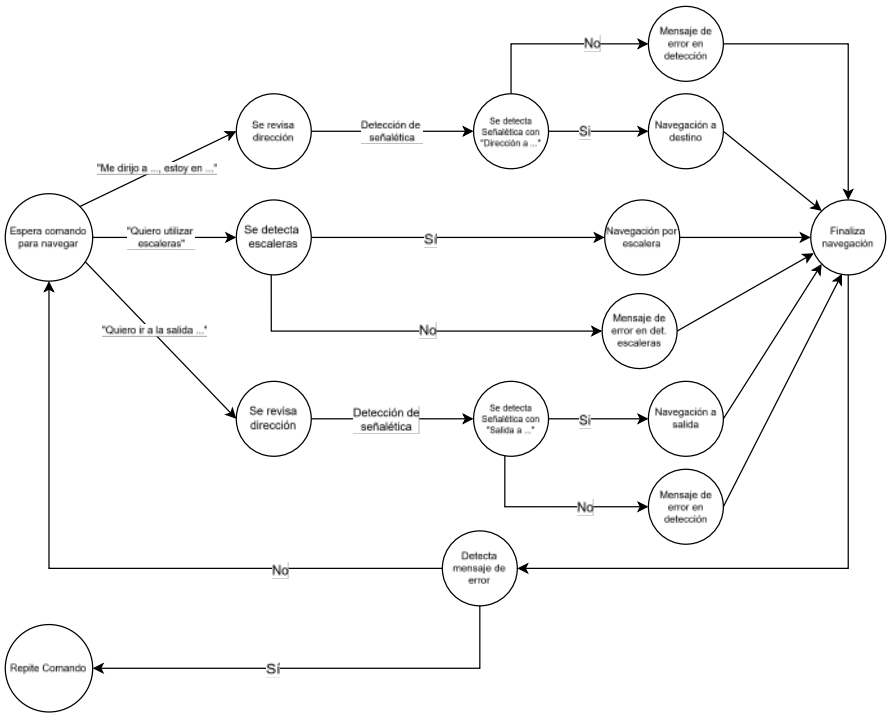


Figura 3.6: *Modelo de navegación: interacción durante la navegación*

Selección de Clases

La definición de las clases a detectar no partió de un criterio abstracto de "elementos relevantes para la orientación", sino que se construyó a partir de las necesidades identificadas durante el levantamiento de requisitos con usuarios, descrito en el Capítulo de Requisitos, y de la propia estructura de señalización del Metro de Santiago.

En primer lugar, la referencia a los pisos podotáctiles responde directamente a lo señalado durante la visita a la Biblioteca Central para Ciegos, donde se destacó que estos elementos constituyen una guía física esencial de desplazamiento dentro de las estaciones, cuya referencia dentro del sistema no resulta redundante sino complementaria. De ahí surgen las clases *Podo Línea* y *Podo Círculo*.

En segundo lugar, los elementos de infraestructura fija de la estación (ascensores, escaleras mecánicas, escaleras normales, y la entrada de torniquetes) corresponden a puntos de decisión obligatorios dentro de cualquier trayecto dentro del Metro, ya sea como medio de desplazamiento vertical o como punto de acceso a la red. La clase *Es-*

tación, correspondiente al logotipo presente en las paredes de los andenes, cumple una función distinta: permite confirmar al usuario que se encuentra efectivamente dentro de una estación determinada, en línea con la necesidad de orientación espacial identificada en la entrevista realizada en la Línea 6, donde se señaló la dificultad para determinar la posición propia dentro del andén.

Finalmente, las seis clases de señalética por color (*Señales Amarillas, Azules, Cafés, Rojas, Rosas y Verdes*) corresponden a la codificación cromática oficial utilizada por Metro de Santiago para identificar cada una de sus líneas. La detección diferenciada por color resulta indispensable para el componente de orientación desarrollado en este trabajo, dado que permite validar, en conjunto con el reconocimiento de texto sobre la señalética detectada, si la dirección hacia la que avanza el usuario corresponde efectivamente a la línea y sentido de viaje esperados.

En conjunto, las 13 clases definidas (resumidas a continuación) cubren tanto los elementos de desplazamiento físico dentro de la estación como los elementos de señalización necesarios para determinar la dirección de viaje del usuario:

1. Ascensores
2. Entrada (entrada de torniquetes a la estación)
3. Escaleras Mecánicas
4. Escaleras Normales
5. Estación (logo de la estación en las paredes de los andenes)
6. Podo Círculo (suelo indicativo de intersección de líneas de pisos podotáctiles)
7. Podo Línea (líneas podotáctiles)
8. Señales Amarillas (línea 2)
9. Señales Azules (línea 4)

10. Señales Cafés (línea 3)
11. Señales Rojas (línea 1)
12. Señales Rosas (línea 6)
13. Señales Verdes (línea 5)

Administración del Dataset

Para la administración del dataset se utiliza la plataforma Roboflow [44], [45], la cual permite centralizar el proceso de carga de imágenes, etiquetado, control de versiones y exportación del mismo hacia formatos compatibles con YOLO.

Además, Roboflow ofrece la ventaja de contar con herramientas integradas de *data augmentation*, que permiten generar variaciones de las imágenes originales ampliando así el número de imágenes disponibles para el entrenamiento. El data augmentation permite exponer al modelo a una mayor variabilidad de condiciones visuales, mejorando su capacidad de generalización frente a escenarios reales dentro de las estaciones del Metro de Santiago, como pueden ser obstrucción parcial visual de los elementos de interés o rotaciones a causa del movimiento del usuario.

3.5.2. Diseño del modelo de detección

Para la detección de señalética y elementos de infraestructura del Metro de Santiago, se opta específicamente por la versión YOLOv11n [46], debido a su balance entre precisión y eficiencia computacional, requisito fundamental considerando que el sistema debe ejecutarse en dispositivos móviles de gama media sin depender de conectividad a internet (procesamiento edge) [47]. Se diseña una etapa comparativa entre tres variantes de arquitectura, con el fin de evaluar qué configuración ofrecía el mejor equilibrio entre precisión y carga computacional:

- **YOLOv11n tradicional**, como línea base.

- **YOLOv11n + capa P2**, agregando una escala adicional orientada a mejorar la detección de objetos pequeños (relevante para elementos como señaléticas distantes o pisos podotáctiles).
- **YOLOv11n + bloques RepViT**, incorporando un backbone inspirado en Vision Transformers para evaluar si el mecanismo de atención mejoraba la robustez frente a variabilidad visual del entorno.

Como criterios de evaluación y selección del modelo final, se definen las siguientes métricas, alineadas con los requisitos no funcionales del sistema: **mAP@0.5** para precisión de detección, **FPS** para velocidad de inferencia (con un umbral mínimo esperado de 20 FPS) y **latencia de respuesta** ($\leq$200ms), con el objetivo de garantizar una experiencia en tiempo real adecuada para la orientación del usuario.

3.5.3. Diseño de la aplicación móvil

La aplicación móvil constituye la plataforma de ejecución del componente de visión por computador. Su función principal dentro de este trabajo consiste en proporcionar acceso a la cámara del dispositivo, ejecutar el modelo de detección de manera local y presentar los resultados obtenidos por el sistema.

La aplicación fue desarrollada para dispositivos Android utilizando Kotlin, considerando como requisito la ejecución local del modelo sin depender de una conexión permanente a Internet. Esta decisión permite que el procesamiento de las imágenes se realice directamente sobre el dispositivo, reduciendo la dependencia de servicios externos y de la disponibilidad de conectividad.

El diseño de la aplicación se estructura en diferentes componentes funcionales, correspondientes a la captura de imágenes, preprocesamiento, inferencia, postprocesamiento y presentación de resultados.

Como complemento al sistema de detección de objetos, se incorporó un componente de reconocimiento óptico de caracteres (OCR), orientado a obtener información

textual presente en el entorno del Metro de Santiago. Mientras que el modelo de detección permite identificar elementos visuales previamente definidos mediante las clases del conjunto de datos, el OCR permite obtener información contenida directamente en textos presentes en señaléticas, carteles u otros elementos del entorno, complementando así la información proporcionada por el detector en situaciones donde el contenido textual de una señal aporta información adicional para la orientación del usuario. El detalle de su funcionamiento interno se desarrolla en el Capítulo de **Implementación**.

3.5.4. Diseño de Interfaz

A continuación, se presenta el diseño preliminar de la interfaz de AIMES, el cual fue desarrollado siguiendo los principios de accesibilidad establecidos por las WCAG 2.1.

El diseño contempla tres versiones iterativas, cada una de las cuales responde a distintos niveles de refinamiento visual y funcional:

La Figura 3.7 muestra una representación esquemática de la **interfaz de usuario** de AIMES, propone una estructura minimalista con botones de gran tamaño, bajo nivel de complejidad y contraste básico.

La Figura 3.8 introduce funciones clave como la visualización de estados y la inclusión de íconos accesibles que refuerzan el contenido textual.



Figura 3.7: *Interfaz Gráfica AIMES*



Figura 3.8: *Interfaz Gráfica AIMES 2*

La Figura 3.9 se perfila como la más robusta visualmente, incorporando un fondo negro, íconos a color de alto contraste y una jerarquía visual clara, lo que favorece su uso por personas con baja visión.

Por otra parte, es importante mencionar que el botón de alerta está diseñado para ofrecer respuestas rápidas y adaptadas a distintas situaciones de urgencia.
Las opciones desarrolladas son las siguientes:



Figura 3.9: *Interfaz Gráfica AIMES 3*

- **Opción 1:** Envío automático de un mensaje predefinido con la ubicación en tiempo real a un contacto previamente configurado, confirmando la acción mediante retroalimentación auditiva.
- **Opción 2:** Incorporación de dos botones, uno para contactar directamente a un contacto predefinido y otro para llamar a servicios de emergencia como policía, bomberos o ambulancias.
- **Opción 3:** Inclusión de mensajes predeterminados con nombre del usuario y ubicación GPS, además de la posibilidad de personalizar mensajes para situaciones de menor urgencia.
- **Opción 4:** Combinación de las funcionalidades anteriores, incorporando 3 botones, contacto predefinido, servicios de emergencia y mensajes predeterminados con interfaz accesible para su selección.

A continuación se muestra en la Figura 3.10 la interfaz gráfica de las opciones propuestas.



Figura 3.10: *Interfaz Gráfica Botón de Alerta*

El tamaño de los íconos fue definido a partir de las recomendaciones de diseño inclusivo, estableciendo un mínimo de 40px para facilitar su visualización, y manteniendo un espaciado adecuado entre elementos interactivos para prevenir errores táctiles.

La elección de colores, por su parte, se fundamentó en la necesidad de asegurar contrastes superiores a 4.5:1, como se observa en la Figura 3.11. Estos valores son recomendados por las WCAG y la selección se centra en evitar combinaciones problemáticas para personas con daltonismo.

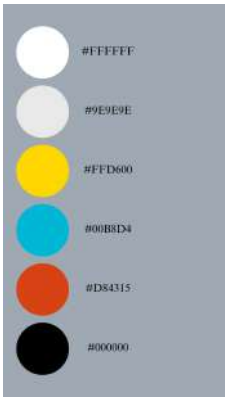


Figura 3.11: *Paleta de colores Interfaz Gráfica AIMES 3*

Además de la interfaz dirigida al usuario final, se ha diseñado una **interfaz provisoria para uso de desarrolladores**, con el objetivo de facilitar el test y validación de los módulos implementados durante las etapas tempranas del proyecto.

Esta interfaz no está orientada a la usabilidad por parte de personas con discapacidad visual, sino a brindar acceso directo a métricas internas clave del sistema.

- **Vista de cámara en tiempo real:** Muestra el video capturado con los *bounding boxes* de los objetos detectados.
- **Métricas de rendimiento visual:** Incluye el número de **FPS**, precisión del modelo (mAP estimado) y clases detectadas por frame.
- **Métricas de rendimiento de voz:** Visualización de la latencia del módulo STT, confirmación de intención y tiempo de respuesta en TTS.

Esta interfaz está implementada dentro de la misma aplicación, pero en una vista distinta accesible solo para test interno. El diseño incluye elementos visuales mínimos, optimizados para revisión técnica más que para experiencia de usuario, como se puede ver en la Figura 3.12.



Figura 3.12: *Interfaz de Desarrollador AIMES*

3.6. Síntesis del diseño y transición a la implementación

El diseño de AIMES fue concebido originalmente como un sistema integral. A partir de las restricciones de tiempo, recursos y las limitaciones técnicas propias de un desarrollo académico, se definió un alcance real acotado, priorizando aquellas funcionalidades con mayor impacto directo sobre la orientación del usuario dentro de la estación.

Sin embargo, incluso dentro de este alcance real, no todas las funcionalidades se lograron completar durante el desarrollo de este trabajo. En particular, el módulo de alertas y la detección de servicios y salidas específicas dentro de la estación (identificados en los requisitos) no alcanzaron a implementarse, quedando como parte del trabajo pendiente del sistema. Tal como se señaló en este capítulo, este diseño establece las bases técnicas y conceptuales sobre las cuales se desarrolla en la fase de implementación, donde se detalla la construcción del dataset, el proceso de entrenamiento y conversión del modelo, y la integración del componente de visión y OCR dentro de la aplicación Android.

4. Implementación de AIMES

En este capítulo se presenta el proceso de implementación de AIMES, describiendo de manera progresiva las distintas etapas que permitieron transformar la solución propuesta en un prototipo funcional para dispositivos móviles. La implementación se estructura en cuatro etapas principales: la construcción del dataset utilizado para el entrenamiento del modelo de visión por computador, el entrenamiento y comparación de arquitecturas basadas en YOLOv11n, la conversión del modelo seleccionado para su ejecución en dispositivos móviles y, finalmente, el desarrollo e integración de la aplicación en Kotlin.

En primer lugar, se describe la construcción del dataset, considerando la recolección inicial de imágenes, su organización y etiquetado mediante la plataforma Roboflow, así como la evolución de las distintas versiones del conjunto de datos hasta alcanzar la configuración utilizada para el entrenamiento final.

A continuación, se presenta el proceso de entrenamiento de YOLOv11n, incluyendo su configuración, las métricas utilizadas para evaluar su desempeño y la comparación entre las distintas arquitecturas consideradas. Una vez seleccionado el modelo, se describe su proceso de conversión a TensorFlow Lite y los intentos de optimización realizados para permitir su integración y ejecución dentro de la aplicación móvil.

Finalmente, se aborda el desarrollo de la aplicación nativa en Kotlin y la integración de los distintos componentes del sistema. Se describe su arquitectura general, la captura y procesamiento de frames, el preprocesamiento de las imágenes, la ejecución de inferencias y la interpretación de las detecciones para estimar la posición relativa de los elementos identificados. Asimismo, se presenta en detalle la incorporación del reconocimiento óptico de caracteres (OCR), utilizado para complementar la detección visual mediante la lectura y validación de información textual presente en la señalética. Por último, se describe la integración de los módulos de reconocimiento y síntesis de voz, que permiten transformar la información procesada por el sistema en indicaciones

auditivas comprensibles para el usuario.

4.1. Construcción de Dataset

4.1.1. Recolección inicial de imágenes

La construcción inicial del dataset se realizó mediante un proceso de captura de imágenes directamente en terreno, con el objetivo de representar las condiciones visuales que podría enfrentar AIMES durante su funcionamiento en el Metro de Santiago. La campaña de adquisición se llevó a cabo en cinco sesiones de captura, realizadas en las siguientes fechas:

- 23 de mayo de 2025.
- 29 de mayo de 2025.
- 25 de junio de 2025.
- 26 de junio de 2025.
- 20 de julio de 2025.

Las capturas se concentraron principalmente en todas las estaciones de la Línea 6 del Metro de Santiago: Cerrillos; Lo Valledor; Pdte. Pedro Aguirre Cerda; Franlink (Combinación con línea 2); Bío Bío; Ñuble (Combinación con línea 5); Estadio Nacional; Ñuñoa (Combinación con línea 3); Inés de Suárez; Los Leones (Combinación con línea 1). Además se complementaron con imágenes obtenidas en estaciones pertenecientes a la Línea 3, específicamente en: Cal y Canto (combinación con Línea 2); Los Libertadores; Ferrocarril; Lo Cruzat; Universidad de Chile (Combinación con línea 1).

Para incorporar señalética correspondiente a las demás líneas de la red, se utilizaron estaciones de combinación, es decir, estaciones en las que convergen dos líneas y donde es posible encontrar señalización asociada a distintos recorridos.

Todas las imágenes empleadas en esta etapa fueron obtenidas directamente durante las visitas a terreno. Para la captura se utilizaron dos dispositivos móviles: un Samsung A33, configurado con una resolución de 12 megapíxeles, y un Huawei Y9 2019, con una cámara de 16 megapíxeles.

El etiquetado fue realizado manualmente por Rodrigo Lobos y Pablo Sánchez mediante la plataforma Roboflow. Para cada elemento de interés se definió una caja delimitadora ajustada a su extensión visible dentro de la imagen, procurando evitar la inclusión innecesaria de regiones correspondientes al fondo. Cuando una imagen contenía más de una instancia perteneciente a una misma clase, cada una fue etiquetada de manera independiente.

La selección de los puntos de captura consideró elementos relevantes para la orientación y el desplazamiento de una persona con discapacidad visual dentro de una estación. Para aumentar la diversidad del conjunto de datos, se registraron imágenes desde distintas posiciones, distancias y perspectivas propias de un recorrido habitual por la red. Entre los escenarios considerados se incluyeron andenes, accesos, recorridos hacia escaleras mecánicas y convencionales, ascensores y elementos presentes en el suelo utilizados como apoyo para el desplazamiento.

Asimismo, se registraron tanto líneas podotáctiles utilizadas para indicar recorridos como superficies podotáctiles de advertencia o cambio de dirección. La incorporación de estos elementos permitió representar características relevantes de la infraestructura accesible del Metro de Santiago y proporcionar ejemplos visuales que posteriormente pudieran ser utilizados por el modelo de detección.

Ejemplos de las clases etiquetadas

El dataset considera trece clases asociadas a elementos de infraestructura y señalética del Metro de Santiago. Seis de ellas corresponden a señaléticas identificadas según el color asociado a las líneas 1, 2, 3, 4, 5 y 6 de Metro. Las Figuras 4.1 y 4.2 presentan ejemplos representativos de estas clases.



(a) *Señal Línea 3 - Color Café* ***(b)*** *Señal Línea 6 - Color Rosa* ***(c)*** *Señal Línea 1 - Color Rojo*

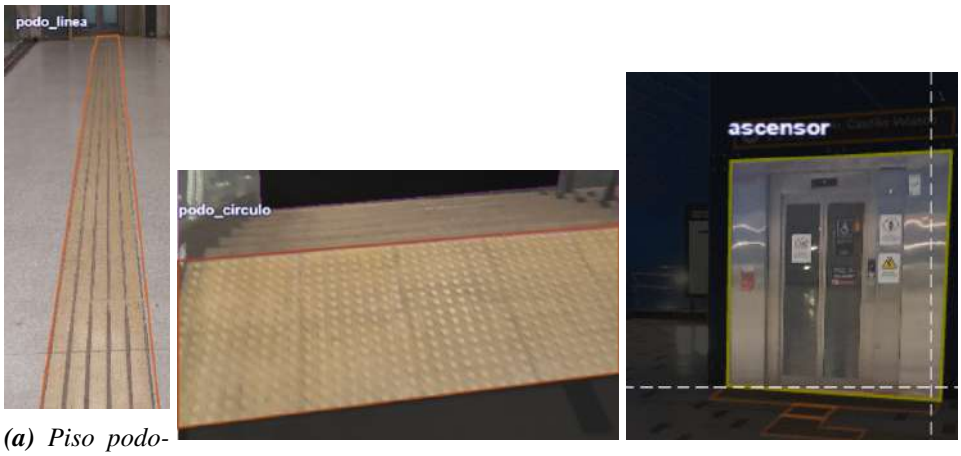
Figura 4.1: *Señales de Metro (1).*



(a) *Señal Línea 4 - Color Azul* ***(b)*** *Señal Línea 2 - Color Amarillo* ***(c)*** *Señal Línea 5 - Color Verde*

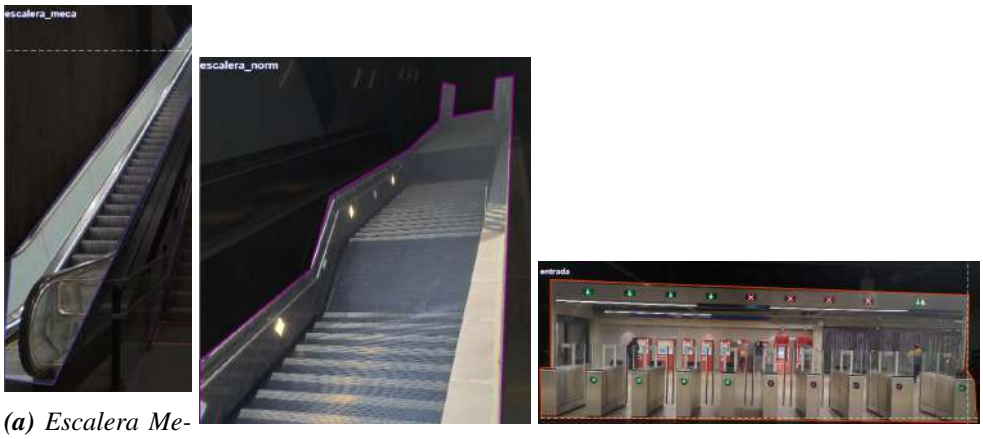
Figura 4.2: *Señales de Metro (2).*

Además de las clases asociadas a señalética, el dataset incorpora elementos de infraestructura relevantes para el desplazamiento dentro de las estaciones, entre ellos pisos podotáctiles, ascensores, entradas y escaleras. Las Figuras 4.3 y 4.4 presentan ejemplos de imágenes etiquetadas correspondientes a estas categorías.



(a) *Piso podotáctil - línea* ***(b)*** *Piso podotáctil - círculo* ***(c)*** *Ascensor*

Figura 4.3: *Infraestructura de Metro (1).*



(a) *Escalera Mecánica* ***(b)*** *Escalera Normal* ***(c)*** *Entrada de estación*

Figura 4.4: *Infraestructura de Metro (2).*

Finalmente, la Figura 4.5 presenta un ejemplo correspondiente a la clase *Estación*, definida para reconocer el identificador visual presente en los andenes del Metro. Esta clase permite detectar la presencia de dicho elemento dentro de la escena, pero no distingue por sí sola el nombre específico de cada estación, ya que todas ellas pertenecen a una misma categoría dentro del modelo.



Figura 4.5: *Estación de Metro*

Las sesiones de captura se realizaron considerando las franjas horarias definidas por Metro de Santiago: **Punta**, comprendida entre las 07:00 y 08:59 horas y entre las 18:00 y 20:44 horas; **Valle**, entre las 09:00 y 17:59 horas; y **Bajo**, entre las 06:00 y 06:59 horas y entre las 20:45 y 23:00 horas.

La mayor parte de las sesiones se realizó durante el período **Valle**. Esta decisión respondió principalmente a razones operativas y de factibilidad para el trabajo en te-

rreno, debido a que dicha franja permitía realizar las jornadas de captura de manera continua y con menores dificultades logísticas. No obstante, incluso durante este período se presentan variaciones en el flujo de pasajeros, lo que permitió registrar escenas con diferentes niveles de ocupación y circulación dentro de las estaciones.

El resultado de esta etapa correspondió a un conjunto inicial de imágenes recopiladas en condiciones reales, que posteriormente fue sometido a procesos de revisión, selección y etiquetado.

Previo a la división en entrenamiento, validación y prueba, se realizó una etapa de curación orientada a eliminar imágenes desenfocadas, capturas sin elementos de interés y secuencias casi idénticas obtenidas de frames consecutivos. Esta separación se realizó antes de aplicar técnicas de data augmentation, con el objetivo de disminuir el riesgo de que imágenes derivadas de una misma captura aparecieran simultáneamente en los conjuntos de entrenamiento y evaluación.

Sobre esta base se desarrollaron las distintas versiones del dataset utilizadas durante las etapas posteriores de entrenamiento y evaluación del modelo de detección.

4.1.2. Organización y Data augmentation en Roboflow

Para la administración del dataset se utilizó la plataforma Roboflow. Sobre el conjunto etiquetado, Roboflow permite además aplicar transformaciones automáticas de *data augmentation* [44], generando múltiples variaciones de cada imagen original antes del entrenamiento. Las técnicas utilizadas a lo largo de las distintas versiones del dataset fueron:

- **Rotación:** gira la imagen dentro de un rango angular definido, manteniendo las anotaciones ajustadas a la nueva orientación.
- **Shear (corte):** aplica una distorsión angular horizontal y/o vertical sobre la imagen, simulando cambios de perspectiva.
- **Cutout:** oculta un número determinado de regiones rectangulares de tamaño de-

finido dentro de la imagen, simulando oclusiones parciales de los objetos de interés.

- **Bounding Box Blur:** aplica un desenfoque de intensidad acotada específicamente sobre el contenido de las cajas delimitadoras, simulando pérdida de nitidez por movimiento de cámara.
- **Ajuste de contraste:** corrige la exposición de la imagen mediante ecualización adaptativa (*Adaptive Equalization*) o expansión de contraste (*Contrast Stretching*), según la versión.

Adicionalmente, en distintas versiones se aplicó un recorte estático (*static crop*) sobre una región definida del encuadre, y un redimensionamiento directo (stretch o deformación directa) de todas las imágenes a una resolución de 640×640 píxeles, compatible con la entrada del modelo a utilizar.

4.1.3. Evolución de versiones del dataset

La construcción del dataset se numera a partir de la versión 16, dado que las versiones anteriores correspondieron a pruebas exploratorias de la plataforma Roboflow y no a un dataset construido en función de los requerimientos del sistema. Entre las versiones 16 y 23 se registran tres etapas diferenciables en la configuración de preprocesamiento y data augmentation aplicado.

Para todas las versiones del dataset (excepto la versión 21 y 22), se generó un augmentation de 3 por cada imagen asociada al entrenamiento (Output/img). Para las versiones 21 y 22, se hizo un augmentation de 2 por cada imagen.

Versiones 16 a 18 (sesiones 1 y 2)

Se mantuvo un conjunto de 1.275 imágenes de entrenamiento (425 imágenes originales), 100 de validación y 40 de prueba (90 %/7 %/3 %), con recorte estático del

25 %-75 % del encuadre y ecualización adaptativa de contraste. La augmentation geométrica se mantuvo constante (rotación entre -15° y +15°, shear de ±10° en ambos ejes), variando únicamente la configuración de Cutout entre versiones: 7 recuadros al 15 % de tamaño (v16), 9 recuadros al 5 % (v17), y 10 recuadros al 6 % (v18).

Versiones 19 y 20 (sesion 3)

El conjunto de datos se amplió a 2.685 imágenes de entrenamiento (895 imágenes originales), 189 de validación y 103 de prueba (90 %/6 %/4 %), eliminando por completo el augmentation de Cutout utilizada previamente. En la versión 20, sobre el mismo volumen de datos, se amplió la región de recorte estático (de 25 %-75 % a 12 %-87 % horizontal, y hasta 25 %-85 % vertical), se reemplazó la ecualización adaptativa por expansión de contraste, y se redujo la intensidad del augmentation (rotación de ±15° a ±6°, shear horizontal de ±10° a ±5°).

Versiones 21 a 23 (sesiones 4 y 5)

Para las versiones 21 y 22, los conjunto se redujo a 1.034 imágenes de entrenamiento (517 imágenes originales), 132 de validación y 73 de prueba (83 %/11 %/6 %), simplificando el preprocesamiento a solo redimensionado (y auto-orientación en v22), sin recorte estático ni ajuste de contraste. En ambas versiones se incorporó desenfoque sobre las cajas delimitadoras (*Bounding Box Blur*) de hasta 2,5 píxeles.

Finalmente, en la versión 23 (base del entrenamiento y comparación de arquitecturas descrita en la siguiente sección) el conjunto de entrenamiento se amplió a 2.237 imágenes de entrenamiento (745 imágenes originales), 132 de validación, y 73 de prueba (92 %/5 %/3 %), combinando una rotación moderada (±6°) con el desenfoque de cajas delimitadoras de la etapa anterior.

La reducción observada en la versión 21 corresponde a una curación del dataset. Durante esta etapa se priorizó la calidad de las imágenes sobre el volumen total, eliminando ejemplos redundantes, capturas de baja calidad y transformaciones que podían

introducir una distorsión excesiva respecto de las condiciones reales de uso.

Posteriormente, el análisis de la distribución de etiquetas evidenció una representación desigual entre las distintas clases del dataset. Algunas clases contaban con una cantidad considerablemente menor de instancias que otras, lo que podía afectar la capacidad del modelo para aprender de manera adecuada los elementos menos representados.

A partir de esta observación, en la versión 23 se amplió nuevamente el conjunto de entrenamiento, incorporando imágenes orientadas principalmente a aumentar la representación de las clases con menor cantidad de etiquetas. De esta manera, el número de imágenes de entrenamiento aumentó de 1.034 a 2.237, buscando disminuir el desbalance entre clases y mejorar la diversidad de ejemplos disponibles para el entrenamiento.

La Tabla 4 resume esta evolución. El análisis del desempeño del modelo asociado a cada una de estas configuraciones, así como los criterios que llevaron a adoptar la versión 23 como base de entrenamiento final, se presentan en el Capítulo de Resultados.

Versión	Train / Val / Test	Cambio principal	Outputs/img
v16	1275 / 100 / 40	Cutout: 7 cajas, 15 %	3
v17	1275 / 100 / 40	Cutout: 9 cajas, 5 %	3
v18	1275 / 100 / 40	Cutout: 10 cajas, 6 %	3
v19	2685 / 189 / 103	Ampliación de dataset; sin Cutout	3
v20	2685 / 189 / 103	Recorte más amplio; menor rotación/shear	3
v21	1034 / 132 / 73	Curación; Bounding Box Blur	2
v22	1034 / 132 / 73	Sin recorte/contraste; Blur mantenido	2
v23	2237 / 132 / 73	Aumento de train para reducir desbalance	3

Tabla 4: *Evolución de versiones del dataset (v16–v23)*

Es importante señalar que los resultados obtenidos entre las etapas v16–v18, v19–v20 y v21–v23 no constituyen una comparación experimental estrictamente controlada, debido a que los conjuntos de validación y prueba fueron modificados durante la evolución del dataset. En consecuencia, las métricas obtenidas entre etapas deben interpretarse como indicadores del proceso iterativo de desarrollo y no como comparaciones directas de rendimiento.

La comparación final entre arquitecturas sí se realizó bajo condiciones homogéneas (modelo de detección), utilizando para todas ellas la misma versión 23 y los mismos conjuntos de validación y prueba.

4.2. Entrenamiento de Modelo

Una vez determinada la versión final del dataset a utilizar, se dio paso al proceso de construcción y entrenamiento del modelo de detección de objetos, etapa central para dotar al sistema de la capacidad de reconocer los elementos de infraestructura y señalética definidos previamente. Esta sección describe la configuración utilizada durante el entrenamiento, las métricas empleadas para evaluar el desempeño de los modelos obtenidos, y la comparación realizada entre distintas variantes de arquitectura sobre la base de YOLOv11n y sus distintas configuraciones.

4.2.1. Configuración de entrenamiento

El entrenamiento de las distintas variantes de arquitectura se realizó sobre la versión final del dataset, descrita en la sección anterior, utilizando imágenes redimensionadas a 640×640 píxeles. Se utilizaron como punto de partida los pesos preentrenados de YOLOv11n sobre el dataset COCO (*Common Objects in Context*), aplicando la técnica de *transfer learning* para adaptar el modelo al dominio específico de señalética e infraestructura del Metro de Santiago. Cada configuración se entrenó durante 100 épocas. El proceso se ejecutó en un equipo local equipado con un procesador AMD Ryzen 3400G y una unidad de procesamiento gráfico NVIDIA GTX 2060 Super, con 16 GB de RAM.

4.3. Métricas de evaluación

Con el fin de seleccionar la arquitectura más adecuada entre las variantes comparadas, se definieron métricas de calidad de detección, calculadas sobre el conjunto de validación durante el proceso de entrenamiento mediante el propio framework de entrenamiento de YOLO.

- **Precisión (Precision):** La Ecuación 1 representa la proporción de detecciones realizadas por el modelo que corresponden efectivamente a un objeto real.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

  donde TP (verdaderos positivos) corresponde a las detecciones correctas, y FP (falsos positivos) a las detecciones erróneas.

- **Recall (Exhaustividad):** La Ecuación 2 representa la proporción de objetos reales que fueron correctamente detectados por el modelo, definida como:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

  donde FN (falsos negativos) corresponde a los objetos presentes en la imagen que el modelo no logró detectar.

- **F1-Score:** La Ecuación 3 representa la media armónica entre precisión y recall, utilizada para resumir ambas métricas en un solo valor cuando existe un compromiso entre ellas:

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

- **Intersection over Union (IoU):** La intersección sobre unión cuantifica el grado de superposición entre la caja delimitadora predicha por el modelo y la caja correspondiente a la anotación real. La Ecuación 4 muestra su cálculo:

$$IoU = \frac{|B_{pred} \cap B_{gt}|}{|B_{pred} \cup B_{gt}|} \tag{4}$$

donde B_{pred} corresponde a la caja predicha y B_{gt} a la caja de referencia.

Average Precision (AP): Para cada clase, la precisión promedio se calcula a partir del área bajo la curva Precision–Recall mostrado en la Ecuación 5.

$$AP_c = \int_0^1 P_c(R)\, dR \tag{5}$$

donde $P_c(R)$ representa la precisión obtenida para la clase c en función del recall.

- **mean Average Precision (mAP):** El mAP corresponde al promedio de los valores de AP obtenidos para las C clases mostrados en la Ecuación 6:

$$mAP = \frac{1}{C} \sum_{c=1}^{C} AP_c \tag{6}$$

- **Non-Maximum Suppression (NMS):** Una vez obtenidas las predicciones del modelo, pueden generarse múltiples cajas superpuestas sobre un mismo objeto. NMS conserva inicialmente la detección de mayor confianza y elimina aquellas cajas cuya superposición con esta supere un determinado umbral de IoU.

Dado que el desbalance entre clases es un riesgo relevante en un dataset construido específicamente para este trabajo, la evaluación no se limitó al valor global de mAP@0.5, sino que se complementó con un análisis por clase, apoyado en tres herramientas estándar generadas durante la validación de cada modelo:

- **Curva Precision-Recall (PR):** permite observar el compromiso entre precisión y recall para cada clase de forma independiente, identificando aquellas clases donde el modelo presenta un desempeño inferior al promedio general.

- **Curva F1-Score por umbral de confianza:** permite determinar el umbral de confianza que maximiza el equilibrio entre precisión y recall, información relevante para calibrar posteriormente el umbral de aceptación de detecciones utilizado en la aplicación móvil.

- **Matriz de confusión:** permite identificar confusiones sistemáticas entre clases visualmente similares (por ejemplo, entre escaleras mecánicas y escaleras normales, o entre distintos colores de señalética bajo condiciones de iluminación desfavorables).

4.4. Conversion Modelo para App Móvil

4.4.1. Exportación a TensorFlow Lite

Una vez entrenadas las tres variantes de arquitectura, los modelos resultantes fueron exportados desde su formato nativo de entrenamiento hacia el formato ONNX (*Open Neural Network Exchange*), como paso intermedio hacia su conversión final. A partir de los archivos ONNX, cada modelo fue convertido a un modelo TensorFlow en formato *SavedModel*, formato requerido por el conversor de TensorFlow Lite, dado que este no admite archivos ONNX de forma directa. Finalmente, cada *SavedModel* fue convertido a formato `.tflite` mediante la utilidad `TFLiteConverter`, sin especificar parámetros adicionales de optimización, generando así un modelo ejecutable localmente dentro de la aplicación Android.

Cabe destacar que las tres arquitecturas comparadas reciben una entrada de $1 \times 3 \times 640 \times 640$ (formato NCHW) y entregan una única salida de $1 \times 17 \times N$, donde las primeras cuatro componentes corresponden a las coordenadas de la caja delimitadora y las trece restantes a la puntuación de cada una de las clases definidas para el sistema. La variante con capa P2 presenta $N = 8500$ anclas de detección, frente a $N = 8400$ en las variantes tradicional y RepViT, diferencia que corresponde a la escala adicional de detección incorporada por dicha capa para mejorar el reconocimiento de objetos de

menor tamaño.

4.4.2. Cuantización y optimización

Durante el proceso de conversión se evaluó la posibilidad de aplicar cuantización a los modelos exportados, con el objetivo de reducir su tamaño y mejorar su velocidad de inferencia en el dispositivo móvil. Sin embargo, al generar variantes cuantizadas del modelo, la aplicación presentó fallos durante la carga e inicialización del intérprete de TensorFlow Lite al momento de ejecutar la inferencia, impidiendo obtener predicciones y provocando el cierre inesperado de la aplicación. Dado que no fue posible resolver esta incompatibilidad dentro de los plazos del proyecto, se optó por descartar la cuantización y mantener los modelos en precisión completa (*float32*), realizando una exportación directa de los pesos originales sin aplicar parámetros adicionales de optimización durante la conversión a TensorFlow Lite. Si bien esta decisión implica un mayor tamaño de modelo y un costo computacional superior al que se obtendría con una versión cuantizada, permitió mantener la funcionalidad completa del sistema dentro de los plazos de desarrollo disponibles. La resolución del problema de compatibilidad identificado y la evaluación del impacto de la cuantización sobre la precisión y el rendimiento del modelo se identifican como líneas de trabajo futuro para optimizar la ejecución de AIMES en dispositivos de gama media.

4.4.3. Validación con benchmark

Para la validación de los modelos exportados se evaluó su eficiencia computacional, considerando como principal referencia el tiempo requerido para realizar la inferencia. Con el objetivo de obtener una comparación homogénea entre las distintas arquitecturas, el proceso de *benchmarking* se ejecutó en el mismo equipo utilizado durante la etapa de entrenamiento, equipado con un procesador AMD Ryzen 5 3400G. La inferencia se realizó exclusivamente mediante CPU, sin utilizar aceleración por GPU, buscando establecer condiciones más representativas de las capacidades de procesamiento

disponibles en un dispositivo móvil de gama media.

La eficiencia computacional se midió directamente sobre los modelos mediante un script de *benchmarking* desarrollado específicamente para este trabajo. La medición se realizó cargando cada modelo con el intérprete de TensorFlow Lite (`tf.lite.Interpreter`), configurado con 4 hilos de CPU, y ejecutándolo sobre el conjunto completo de imágenes de prueba del dataset.

Previo a la medición, se ejecutaron 10 inferencias de calentamiento (*warm-up*) sobre el intérprete, con el fin de excluir del cálculo el costo de inicialización del modelo. Posteriormente, cada imagen fue procesada 3 veces consecutivas, midiendo el tiempo de cada inferencia mediante un temporizador de alta resolución (`time.perf_counter_ns`), desde la asignación del tensor de entrada (`set_tensor`) hasta la obtención completa de los tensores de salida (`get_tensor`), convirtiendo el resultado de nanosegundos a milisegundos.

A partir de estas mediciones se derivaron dos métricas de velocidad, correspondientes a distintos alcances de medición:

- **FPS de Inferencia:** La Ecuación 7 muestra el cálculo asociado a los FPS de inferencia.

$$\mathrm{FPS}_{\mathrm{inf}} = 1000 \,/\, \bar{t}_{\mathrm{inferencia}} \tag{7}$$

  Donde $\bar{t}_{\mathrm{inferencia}}$ corresponde al tiempo promedio de inferencia en milisegundos. Esta métrica refleja exclusivamente la velocidad del modelo en sí, sin considerar el tiempo de lectura o preprocesamiento de las imágenes.

- **FPS global:** calculado como el número total de inferencias realizadas dividido por el tiempo total transcurrido durante todo el proceso de benchmarking, incluyendo la carga y el preprocesamiento de cada imagen. Esta métrica aproxima de mejor manera el rendimiento esperado en un escenario de uso continuo.

Adicionalmente, se calcularon estadísticas descriptivas del tiempo de inferencia (media, mediana, percentil 95, valor mínimo y máximo), con el fin de caracterizar

no solo el desempeño promedio de cada modelo, sino también su estabilidad frente a valores atípicos que pudiesen afectar la experiencia del usuario en tiempo real.

En paralelo a la medición de tiempos, se ejecutó un hilo independiente de monitoreo de recursos mediante la librería `psutil`, que registró el uso de CPU y memoria RAM del proceso cada 50 ms durante todo el benchmark. El uso de CPU se normalizó respecto al número de núcleos lógicos disponibles, de modo que un valor de 100 % representa la utilización completa de un núcleo. A partir de estas muestras se obtuvo el uso promedio y máximo de CPU y RAM para cada modelo, junto con el tamaño en disco de cada archivo `.tflite` correspondiente a cada variante.

En conjunto, estas métricas (FPS de inferencia, FPS global, latencia, uso de CPU, uso de RAM y tamaño del modelo) se contrastan con los umbrales definidos como requisito no funcional del sistema (mínimo de 20 FPS y latencia máxima de 200 ms), constituyendo el criterio de eficiencia computacional utilizado, en conjunto con las métricas de calidad de detección de la sección anterior, para la selección final del modelo. Los resultados obtenidos para cada variante se presentan en el Capítulo de Resultados.

4.5. Desarrollo de Aplicación en Kotlin

Tal como se dijo en la fase de diseño, la implementación de la aplicación se lleva a cabo de forma nativa en Android, debido a las limitaciones que se presentan con el framework de Kivy/Buildozer. A continuación se presenta el desarrollo por etapas de la aplicación.

4.5.1. Entorno de desarrollo de aplicación

La Tabla 5 presenta todas las tecnologías para el desarrollo de la aplicación, desde la construcción de interfaz, hasta el uso de las funcionalidades de AIMES.

Tecnología	**Uso dentro de AIMES**
Android Studio	Entorno utilizado para compilar, depurar y desplegar la aplicación nativa en dispositivos Android.
Kotlin	Lenguaje principal para implementar la lógica de la aplicación, módulos visuales, orientación e interacción por voz.
Jetpack Compose	Construcción de la interfaz móvil, pantallas auxiliares, navegación y vista técnica de prueba.
CameraX	Captura de cámara, vista previa y análisis de frames en tiempo real.
TensorFlow Lite	Ejecución local del modelo de detección en formato `.tflite`.
SpeechRecognizer	Reconocimiento de comandos de voz del usuario mediante APIs nativas de Android.
TextToSpeech	Generación de instrucciones y confirmaciones auditivas.
OCR	Detección de texto para determinar navegación y mejorar retroalimentación

Tabla 5: *Lista de tecnologías utilizadas en la implementación de AIMES.*

Como se puede ver en la tabla, dentro de la implementación de AIMES existen 4 puntos o dependencias: interfaz móvil, captura y procesamiento visual, reconocimiento óptico de caracteres (OCR) e interacción por voz.

Para el desarrollo de la aplicación se emplearon diversos componentes orientados a cubrir las distintas etapas del flujo de funcionamiento del sistema. Como base de la interfaz visual se utilizó principalmente Jetpack Compose, aprovechando sus compo-

nentes para la construcción de pantallas, gestión de estados visuales y navegación entre vistas. Esto permitió desarrollar tanto la interfaz principal de interacción con el usuario como una pantalla destinada a pruebas y depuración, en la cual es posible visualizar la imagen proveniente de la cámara, las detecciones realizadas por el modelo y distintas métricas internas asociadas al proceso de inferencia.

Para la captura y procesamiento de imágenes se incorporaron las bibliotecas de CameraX, encargadas de gestionar la vista previa de la cámara y el análisis de los frames capturados. Estas herramientas resultaron fundamentales para separar el flujo de visualización del flujo de procesamiento, permitiendo mantener una representación continua del entorno mientras, de manera paralela, se ejecutan las tareas de detección y reconocimiento.

En cuanto a la detección de elementos del entorno, se integró TensorFlow Lite para ejecutar el modelo YOLOv11 directamente en el dispositivo móvil. Esta dependencia permite cargar el modelo almacenado en los recursos de la aplicación, preparar los tensores de entrada y procesar sus salidas para obtener las detecciones correspondientes. Los resultados obtenidos son posteriormente interpretados utilizando un archivo de etiquetas asociado, que permite transformar las salidas numéricas del modelo en las clases definidas para el sistema.

Un componente adicional corresponde al reconocimiento óptico de caracteres (OCR), incorporado para procesar la información textual presente en el entorno del usuario. Este módulo permite analizar regiones de interés obtenidas a partir de las imágenes capturadas por la cámara y extraer el texto contenido en ellas. La información reconocida es posteriormente procesada y normalizada para facilitar su comparación con el destino proporcionado por el usuario.

Finalmente, la interacción por voz se implementó principalmente mediante las APIs nativas de Android como `SpeechRecognizer`, utilizado para recibir instrucciones del usuario, mientras que la retroalimentación auditiva se generó mediante `TextToSpeech`. De esta forma, la interacción por voz permite establecer el destino y recibir las instruc-

ciones generadas por el sistema, integrándose con la información obtenida mediante la detección visual y el reconocimiento de texto.

4.5.2. Arquitectura de la aplicación

La aplicación Android de AIMES se estructura en un conjunto de componentes especializados, cada uno responsable de una etapa específica del flujo de asistencia, comunicados entre sí de forma directa mediante llamadas a funciones y observación de estado. La Figura 4.6 presenta esta arquitectura general.

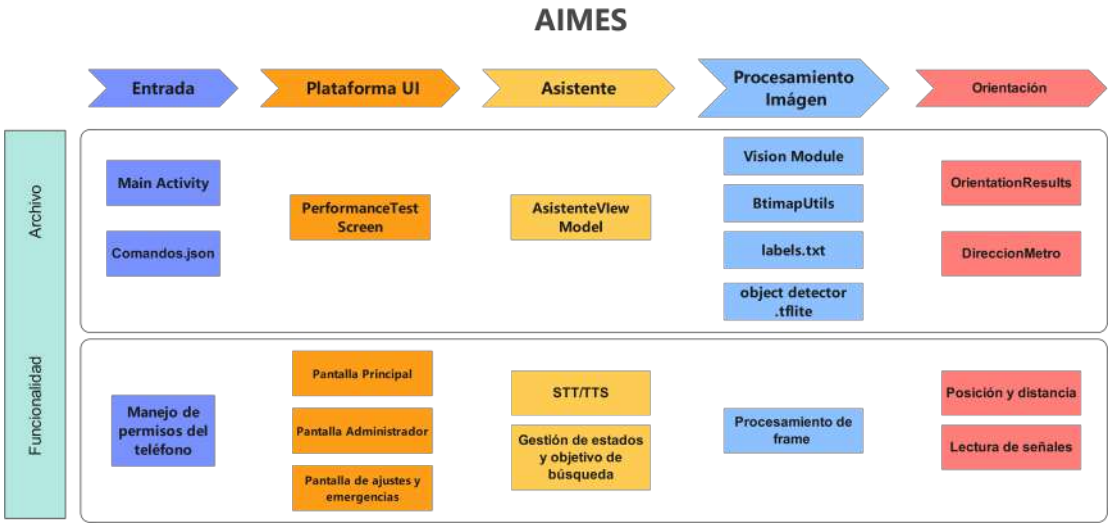


Figura 4.6: *Arquitectura de código de la aplicación AIMES implementada en Android.*

- **`MainActivity`:** gestiona los permisos de uso (cámara y micrófono) y da inicio a los dos flujos principales del sistema, dirigiendo al usuario hacia `AsistenteViewModel` o hacia `PerformanceTestScreen` según corresponda.
- **`AsistenteViewModel`:** centraliza el reconocimiento y síntesis de voz, apoyándose en el archivo `comandos.json` para interpretar la intención del usuario a partir de sus instrucciones.

- **PerformanceTestScreen:** coordina el pipeline visual, delegando en `VisionModule` la ejecución del modelo de detección sobre cada frame capturado por la cámara.
- **VisionModule:** ejecuta el modelo de detección sobre los frames capturados, entregando las detecciones resultantes a `PerformanceTestScreen` y, posteriormente, a `OrientationResults`.
- **OrientationResults:** interpreta las detecciones recibidas para estimar la posición del usuario y, cuando corresponde, valida su dirección mediante reconocimiento óptico de caracteres.
- **DireccionMetro:** es consultado por `OrientationResults` para determinar hacia qué extremo de la línea debe dirigirse el usuario, según la estación de origen y destino declaradas.

Finalmente, tanto `PerformanceTestScreen` como `OrientationResults` retroalimentan a
`AsistenteViewModel`, cerrando el ciclo mediante la generación de instrucciones habladas mediante *text-to-speech*. Esta arquitectura separa claramente tres responsabilidades: la interacción por voz, el procesamiento visual y la lógica de orientación/navegación, permitiendo que cada componente sea desarrollado y probado de forma relativamente independiente.

La Figura 4.7 presenta el diagrama de interacción entre los componentes de la aplicación AIMES, detallando la dirección y el tipo de información que se intercambia entre ellos durante un ciclo completo de asistencia.

El flujo se origina en `MainActivity`, responsable de solicitar los permisos de cámara y micrófono necesarios para el funcionamiento del sistema, y de dirigir al usuario hacia los dos módulos principales de interacción: `AsistenteViewModel`, al cual transmite los comandos de voz recibidos, y `PerformanceTestScreen`, hacia el cual habilita la vista de cámara. `AsistenteViewModel` actúa como intermediario de estado entre `MainActivity` y `PerformanceTestScreen`: le transmite actualizaciones de

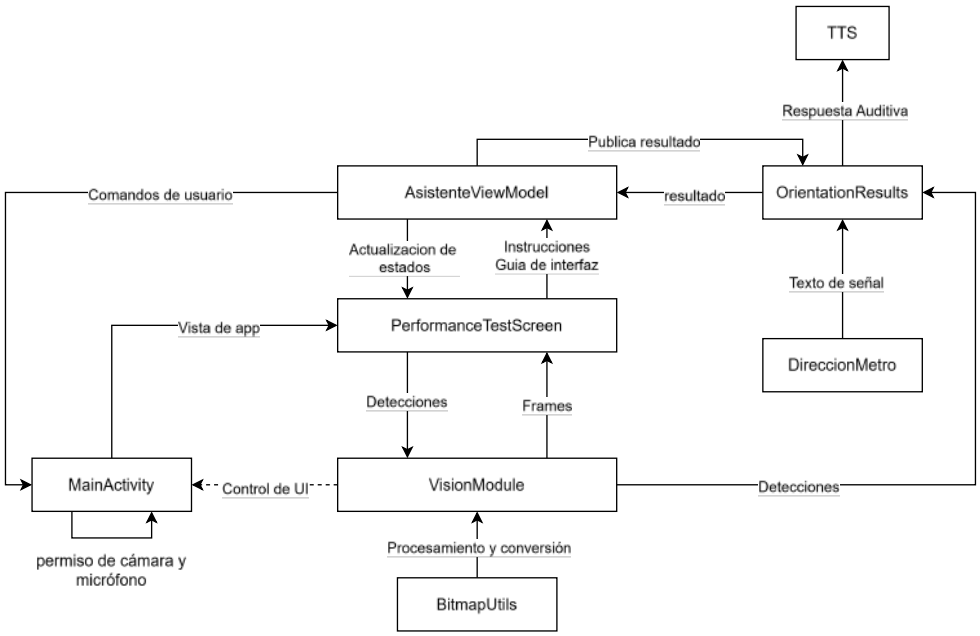


Figura 4.7: *Diagrama de flujo de la aplicación AIMES.*

estado (por ejemplo, la estación de origen y destino indicadas por el usuario), mientras que `PerformanceTestScreen` le retorna instrucciones para guiar la interfaz.

Dentro del pipeline visual, `PerformanceTestScreen` envía los frames capturados por la cámara a `VisionModule`, el cual (apoyándose en `BitmapUtils` para el procesamiento y conversión de imágenes) retorna el conjunto de detecciones identificadas en cada frame. Estas detecciones se envían tanto de vuelta a `PerformanceTestScreen`, para su representación visual, como a `OrientationResults`, encargado de interpretar dichas detecciones en términos de posición y dirección.

Para validar la dirección de viaje, `OrientationResults` se apoya en `DireccionMetro`, que le entrega el texto de señalética esperado según la estación de origen y destino previamente declaradas por el usuario. A partir de esta validación, `OrientationResults` publica el resultado final hacia `AsistenteViewModel`, quien lo traduce en una respuesta auditiva para el usuario, cerrando así el ciclo de retroalimentación iniciado por el comando de voz inicial.

4.5.3. Captura y procesamiento de imagen

La captura de video se implementa mediante la librería CameraX, utilizando dos casos de uso en paralelo: un flujo de *Preview* destinado exclusivamente a la visualización en pantalla, y un flujo de *ImageAnalysis* destinado al procesamiento por el modelo de detección. Este segundo flujo se configura con la estrategia de contrapresión `STRATEGY_KEEP_ONLY_LATEST`, la cual descarta frames pendientes de análisis cuando el procesamiento del frame anterior aún no ha finalizado, priorizando la actualidad de la información entregada al usuario por sobre el procesamiento exhaustivo de la totalidad de frames capturados por la cámara. El análisis de cada frame se ejecuta en un hilo dedicado, independiente del hilo principal de la interfaz.

La Figura 4.8 muestra el procesamiento que tiene la captura de las imágenes en la aplicación, hasta el resultado final en el cual se basa la aplicación para entregar retroalimentación con esta información procesada.

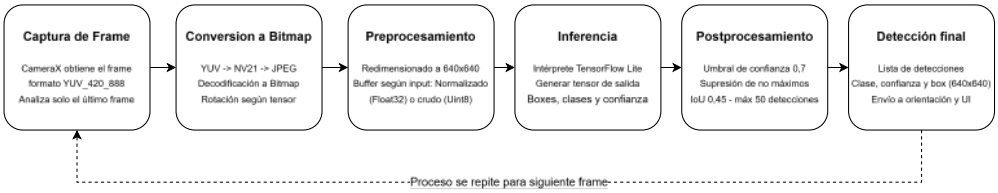


Figura 4.8: *Flujo de Procesamiento de Frame.*

Conversión de formato

Cada frame capturado por CameraX se entrega en formato YUV_420_888, compuesto por planos separados de luminancia (Y) y crominancia (U, V). Dado que el modelo de detección requiere una imagen en formato de mapa de bits (*Bitmap*), se implementó una función de conversión (`BitmapUtils`) que combina los tres planos en un arreglo NV21, lo codifica como una imagen JPEG en memoria, y posteriormente lo decodifica nuevamente como un objeto `Bitmap`. Adicionalmente, dado que la orientación física del sensor de la cámara no coincide necesariamente con la orientación de

visualización del dispositivo, se aplica una rotación a la imagen resultante de acuerdo con el ángulo reportado por el propio sistema de cámara para cada frame.

Preprocesamiento para el modelo

Una vez obtenido el *Bitmap* correctamente orientado, la imagen se redimensiona a una resolución fija de 640×640 píxeles, coincidente con la entrada esperada por el modelo YOLOv11n exportado a TensorFlow Lite. A partir de esta imagen redimensionada, se construye el buffer de entrada del intérprete de TensorFlow Lite, cuyo formato depende del tipo de dato esperado por el modelo: si la entrada es de tipo `FLOAT32`, cada canal de color se normaliza a un rango de 0 a 1; si la entrada es de tipo `UINT8`, los valores de color se entregan directamente en su rango original de 0 a 255. Esta verificación permite que el mismo componente sea compatible con distintas variantes de exportación del modelo, independientemente de si estas incorporan o no cuantización.

Obtención de las detecciones

Tras ejecutar la inferencia, el modelo entrega un tensor de salida **T** [48], La Ecuación 8 representa matemáticamente el tensor, donde 1 corresponde al tamaño del lote (*batch*), C representa la cantidad de atributos asociados a cada candidato de detección (cuatro coordenadas de caja más las trece clases definidas para el sistema) y K corresponde a la cantidad de anclas de detección evaluadas por el modelo.

$$\mathbf{T} \in \mathbb{R}^{1\times C\times K} \tag{8}$$

La Ecuación 9 representa la clase predicha para cada candidato k, donde $p_{k,c}$ corresponde a la puntuación asignada por el modelo al candidato k para la clase c.

$$c_k = \arg\max_{c} \; p_{k,c} \tag{9}$$

La Ecuación 10 representa la confianza para dicho candidato, en donde se elige el

valor máximo de puntuación asignada entre todos estos.

$$s_k = \max_c \; p_{k,c} \tag{10}$$

A partir de esta representación, se conservan únicamente los candidatos cuya confianza s_k supera el umbral de 0,70, se transforman sus coordenadas desde su representación normalizada (c_x, c_y, w, h) hacia un rectángulo delimitador en el espacio de 640×640 píxeles del modelo.

Debido a que un mismo objeto puede generar múltiples detecciones superpuestas sobre anclas vecinas, se aplica a continuación una etapa de supresión de no máximos (*Non-Maximum Suppression*, NMS), descartando aquellas detecciones cuya intersección sobre unión (IoU) respecto a una detección de mayor confianza supera un umbral de 0,45. Las detecciones resultantes se ordenan por confianza descendente, conservando un máximo de 50 detecciones por frame.

Resultado del procesamiento

El resultado final de este pipeline corresponde a una lista de objetos detectados en el frame actual, cada uno compuesto por su clase identificada, su nivel de confianza y su posición dentro del espacio de 640×640 píxeles del modelo. Junto con esta lista, el componente calcula dos indicadores en tiempo de ejecución: la cantidad de frames procesados por segundo (FPS), actualizada una vez por segundo, y la confianza promedio de las detecciones presentes en el frame actual. Este segundo indicador corresponde al promedio de los niveles de confianza asignados por el modelo a las detecciones aceptadas y no debe confundirse con la métrica de precisión utilizada durante la evaluación del modelo. Este conjunto de información (detecciones, FPS y confianza promedio) se entrega mediante una función de retorno (*callback*) a `PerformanceTestScreen`, que lo utiliza tanto para actualizar la representación visual en pantalla como para reenviarlo al módulo de orientación, descrito en la siguiente sección.

4.5.4. Interpretación y orientación de dirección

Una vez obtenida la lista de detecciones para un frame determinado, `OrientationResults` las transforma en información espacial interpretable: para cada detección, se calcula su posición relativa dentro del frame y su distancia aproximada respecto al usuario.

Estimación de posición y distancia

La posición horizontal de cada objeto se determina dividiendo el ancho del frame en cinco franjas iguales según la coordenada central de su caja delimitadora, clasificándola como *izquierda*, *centro-izquierda*, *centro*, *centro-derecha* o *derecha*.

La distancia aproximada se estima a partir de la altura relativa de la caja respecto a la altura total del frame (un objeto que ocupa una mayor proporción vertical del frame se interpreta como más cercano), clasificándose en tres niveles: *cerca* (altura relativa igual o superior a 0,45), *media distancia* (igual o superior a 0,25) y *lejos* (por debajo de dicho umbral).

Selección del objeto relevante

Cuando el módulo opera con una clase objetivo definida (por ejemplo, cuando el usuario solicitó explícitamente orientación hacia escaleras o una entrada), se filtran las detecciones correspondientes a dicha clase y se selecciona aquella con mayor confianza; en ausencia de una clase objetivo, se selecciona la detección de mayor confianza entre la totalidad de objetos identificados en el frame.

Retroalimentación auditiva ambiental

A partir del objeto seleccionado, se genera una frase de retroalimentación auditiva que combina su identidad, posición y distancia (por ejemplo, “escaleras a la derecha, cerca, 82 porciento”), incluyendo el nivel de confianza de la detección. Para evitar una

sobrecarga de información auditiva, se aplica una regla de supresión específica para los elementos de piso podotáctil: dado que su detección es prácticamente continua mientras el usuario se desplaza sobre o cerca de ellos, solo se anuncian cuando se encuentran lejos del usuario, evitando repetir el mismo aviso en cada frame mientras el usuario transita sobre dicha guía física.

Estimación de dirección de giro

De forma complementaria a la retroalimentación ambiental, el módulo implementa un mecanismo de inferencia de dirección basado en la posición horizontal de un letrero respecto al centro del frame: si dicha posición se desvía del centro más allá de un umbral del 15 %, se infiere una instrucción de girar a la izquierda o a la derecha; en caso contrario, se infiere continuar en línea recta. A este umbral se le añade un margen adicional de histéresis del 5 %, con el fin de evitar cambios bruscos entre instrucciones cuando la posición del letrero se encuentra oscilando cerca del límite de decisión. Esta estimación geométrica constituye la base del mecanismo de guiado hacia el destino, cuya activación (condicionada a la validación del letrero correcto mediante OCR) se detalla en el siguiente párrafo.

4.5.5. OCR y confirmación de ruta

A diferencia de la interpretación geométrica descrita en la subsección anterior, el componente OCR no responde a “dónde está el letrero”, sino a “hacia dónde indica ese letrero llevarme”. Esta distinción es relevante porque las seis clases de señalética por color detectadas por el modelo de visión identifican únicamente la línea del Metro a la que pertenece una señalética, pero no permiten, por sí solas, determinar hacia cuál de los dos extremos de esa línea debe dirigirse el usuario. Dicha información solo está disponible como texto impreso en la propia señalética (por ejemplo, “Dirección a Los Leones”), lo que hace necesaria la incorporación de un componente de reconocimiento óptico de caracteres para completar la validación de ruta.

Condiciones de activación

El proceso de validación mediante OCR (`guideWithOcr`) solo se ejecuta cuando existe una imagen de previsualización disponible y el usuario ha declarado tanto una estación de origen como una de destino. Adicionalmente, se aplican dos controles para evitar ejecuciones innecesarias o superpuestas: un intervalo mínimo de 5 segundos entre ejecuciones sucesivas (*cooldown*), y una bandera de ocupación (`ocrBusy`) que impide iniciar un nuevo ciclo de reconocimiento mientras uno anterior aún se encuentra en curso, dado que el reconocimiento de texto es una operación asíncrona que no necesariamente finaliza antes de que llegue un nuevo frame de cámara.

Selección de regiones candidatas

De la totalidad de detecciones del frame actual, se filtran únicamente aquellas correspondientes a las seis clases de señalética por color, y se seleccionan hasta las tres detecciones de mayor área. Priorizar por área responde a que una señal más grande dentro del frame corresponde, en general, a una señal más cercana o mejor encuadrada, y por lo tanto con mayor probabilidad de que su texto resulte legible para el reconocedor óptico.

Lectura y validación del texto

Cada recorte se procesa de forma secuencial mediante Google ML Kit, utilizando el reconocedor de texto latino (`TextRecognizerOptions.DEFAULT_OPTIONS`). Antes de comparar el texto reconocido, tanto este como el nombre de la estación esperada se normalizan, eliminando tildes y unificando la letra “ñ”, con el fin de reducir el impacto de variaciones menores introducidas por el propio proceso de reconocimiento. La validación no exige una coincidencia exacta: se considera válida si el texto normalizado de la estación está contenido por completo en el texto reconocido, o si al menos sus primeros cinco caracteres aparecen dentro de él (permitiendo, por ejemplo, reconocer “cerri” dentro de un texto ruidoso como “direccion cerrilos”). La estación esperada no

se determina arbitrariamente, sino que se obtiene consultando a `DireccionMetro` con la estación de origen y destino declaradas por el usuario, la cual determina hacia cuál de los dos extremos de la línea corresponde dirigirse. El proceso se detiene en el primer recorte cuyo texto reconocido valida esta coincidencia, sin continuar evaluando los recortes restantes.

La Figura 4.9 muestra el proceso gráfico de la lectura y validación:

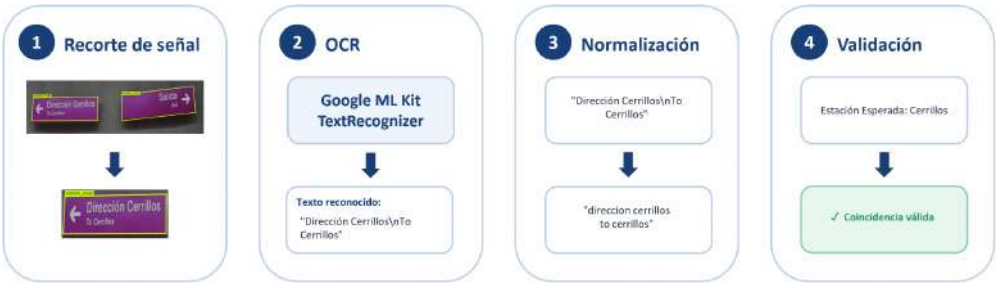


Figura 4.9: *Proceso de OCR: Selección de objeto de interés; Reconocimiento de texto; Normalización de caractéres; Coincidencia con estación esperada*

Cambio de inferencia al confirmar el destino

Mientras ninguna de las señales evaluadas confirma el texto esperado, el sistema no cuenta con una instrucción de navegación válida, dado que la sola posición geométrica de un letrero no garantiza que este corresponda a la dirección correcta. Una vez que el texto reconocido en alguna de las señales candidatas confirma la estación esperada, el flujo de inferencia cambia: se calcula una instrucción de giro a partir de la posición geométrica de esa señal específica (de la misma forma descrita en la subsección anterior), la cual puede ser inmediatamente sobrescrita si el propio texto reconocido contiene una flecha o una palabra direccional explícita. De esta manera, la validación del destino mediante OCR no solo habilita la generación de una instrucción de navegación, sino que también determina cuál de las dos fuentes de información (geometría o texto) prevalece al formular dicha instrucción. La frase resultante se entrega por voz únicamente si el tipo de instrucción (seguir recto, girar a la izquierda o girar a la derecha) difiere de la

última instrucción emitida, evitando repeticiones mientras el usuario se aproxima a la misma señal.

Manejo del caso sin coincidencia

Si ninguna de las señales evaluadas en el ciclo actual logra validar el texto esperado, el sistema no permanece en silencio: se emite un mensaje auditivo indicando que la dirección del andén es incorrecta, señalando explícitamente hacia qué extremo de la línea debe dirigirse el usuario según lo determinado por `DireccionMetro`. Este mecanismo permite que el usuario reciba una indicación correctiva incluso cuando la señalética detectada no corresponde a la dirección de viaje deseada.

Integración con otros componentes

El componente OCR se apoya en la información producida por los módulos descritos previamente: recibe las detecciones y la imagen de previsualización desde `VisionModule`, a través de
`PerformanceTestScreen`; consulta a `DireccionMetro` para determinar la estación esperada a partir del estado de origen y destino gestionado por `AsistenteViewModel`; y finalmente entrega su resultado (ya sea la instrucción de navegación confirmada o el mensaje de corrección de dirección) a este último, quien lo traduce en una respuesta auditiva para el usuario.

4.5.6. Comandos de voz y feedback de AIMES

La interacción por voz en AIMES se estructura como un ciclo continuo de escucha, interpretación y respuesta, gestionado por `AsistenteViewModel`: al finalizar cada respuesta hablada, el reconocimiento de voz se reinicia automáticamente, permitiendo encadenar instrucciones sin volver a presionar el micrófono.

Reconocimiento e interpretación

El reconocimiento se implementa mediante la API nativa `SpeechRecognizer`, configurada en español de Chile con reconocimiento de forma libre. Cada texto reconocido se interpreta en orden de prioridad: primero se evalúa si declara una estación de origen (por ejemplo, "estoy en Los Leones"), mediante expresiones regulares; si no, se contrasta contra los comandos definidos en `comandos.json`, cada uno con múltiples variantes de redacción para una misma intención. La coincidencia no exige una frase exacta: tanto el texto del usuario como cada variante se normalizan y descomponen en conjuntos de palabras, aceptando la coincidencia si todas las palabras de alguna variante están presentes en el texto reconocido, independientemente del orden. Si ninguna vía anterior aplica y el sistema espera un destino, se intenta extraerlo mediante patrones adicionales (por ejemplo, "voy a...", "destino..."); en caso contrario, se solicita al usuario repetir la instrucción.

Ejecución de acciones y estado compartido

Cada intención identificada se traduce en una acción concreta: fijar origen y solicitar destino (`ubicar`), fijar destino e iniciar guiado (`destino_estacion`), declarar un objeto a buscar (`buscar_objeto`), activar un modo de guiado, o gestionar comandos de control como confirmar, cancelar o repetir la última respuesta. Estas acciones actualizan un estado compartido (estación actual, destino, objeto buscado y modo de guiado) consumido por `PerformanceTestScreen` y, a través de ella, por `OrientationResults` y `DireccionMetro`, conectando la interacción por voz con el resto de los componentes descritos anteriormente.

Retroalimentación por voz La síntesis se implementa mediante `TextToSpeech` en español de Chile, reemplazando de inmediato cualquier locución en curso ante una nueva respuesta. Esta misma función expone el método `say()`, invocado por `OrientationResults` para comunicar tanto las instrucciones de navegación confir-

madas mediante OCR como los mensajes de corrección de dirección, consolidando a `AsistenteViewModel` como el único punto de salida auditiva del sistema.

4.5.7. Interfaz Móvil

La interfaz de AIMES fue implementada en Android Studio utilizando Jetpack Compose, siguiendo como referencia el diseño inicial correspondiente tanto a la vista de menú principal como a la vista de desarrollador presentadas en el Capítulo de Diseño. Debido a las limitaciones y restricciones propias del entorno de desarrollo nativo de Android, no fue posible replicar con exactitud dicho diseño original, por lo que se realizó una adaptación visual que conserva la estructura y los elementos principales de la propuesta inicial.

Menú Principal

Tomando como referencia el diseño de la Figura 3.9, se obtuvo el resultado mostrado en la Figura 4.10, desarrollado en Android Studio.



Figura 4.10: *Pantalla Principal de AIMES*

En esta pantalla se observa la aplicación del requisito de accesibilidad visual defi-

nido en el Capítulo de Requisitos, priorizando un alto contraste de colores y elementos de gran tamaño, orientados a usuarios con baja visión.

Emergencias

Como se presentó en la Figura 4.11, se desarrolló la vista de Emergencias, adoptando el enfoque de la opción 4 discutida en el diseño de la solución, por constituir un equilibrio entre la generación de un mensaje de alerta orientado a obtener apoyo de terceros físicamente cercanos, y las opciones de contacto directo con contactos de emergencia y servicios de emergencia formales. Esta pantalla permite al usuario activar dicha alerta desde el menú principal, sin requerir una secuencia adicional de navegación, dada la naturaleza urgente de su propósito.



Figura 4.11: *Pantalla de Emergencia AIMES*

Interfaz de Desarrollador

Como se observa en la Figura 3.12, la pantalla “Prueba de rendimiento de visión” destina la parte inferior de la interfaz a la visualización de métricas en tiempo real. Mientras el sistema no ha identificado ningún objeto relevante, se muestra el indicador de estado “Buscando objetivo…” sobre el área de previsualización de la cámara; al producirse una detección, este indicador es reemplazado por la etiqueta de la clase detectada, la cual se despliega en pantalla al mismo tiempo que se entrega como retroalimentación auditiva mediante el mecanismo de descripción ambiental descrito en la Sección de Interpretación y Orientación de Dirección. De esta forma, la interfaz de desarrollador no solo permite supervisar visualmente el desempeño del sistema, sino que refleja en pantalla la misma información que el usuario final recibe por voz.

Junto a este indicador se exhiben, en paralelo, las métricas de FPS y confianza promedio de detección. La Figura 4.12 presenta esta vista, la cual incorpora además dos controles principales: el botón “Iniciar prueba”, utilizado para dar inicio al ciclo de detección, y un botón de micrófono acompañado de una etiqueta de estado (“Mic apagado” o su equivalente al estar activo), que indica si el sistema se encuentra escuchando comandos de voz. Finalmente, en la parte inferior de la pantalla se incorpora un campo dedicado a mostrar el texto reconocido por el componente OCR (“OCR: ...”), permitiendo verificar directamente, durante las pruebas, el contenido leído desde la señalética detectada.



Figura 4.12: *Pantalla de Desarrollador AIMES*

5. Resultados

5.1. Dataset para el Modelo

5.1.1. Distribución de Etiquetas

La Figura 5.1 representa la distribución de etiquetado de esta versión

Esta distribución no es arbitraria, sino que responde en parte a una decisión deliberada de priorización durante la recolección de imágenes. La clase *Señales Rosas*, correspondiente a la Línea 6 del Metro de Santiago, concentra la mayor cantidad de instancias etiquetadas (990), muy por sobre el resto de las señales de color. Esta priorización responde a que la Línea 6 fue definida como el entorno principal de validación del sistema, tanto por corresponder a la línea donde se realizó la entrevista en terreno

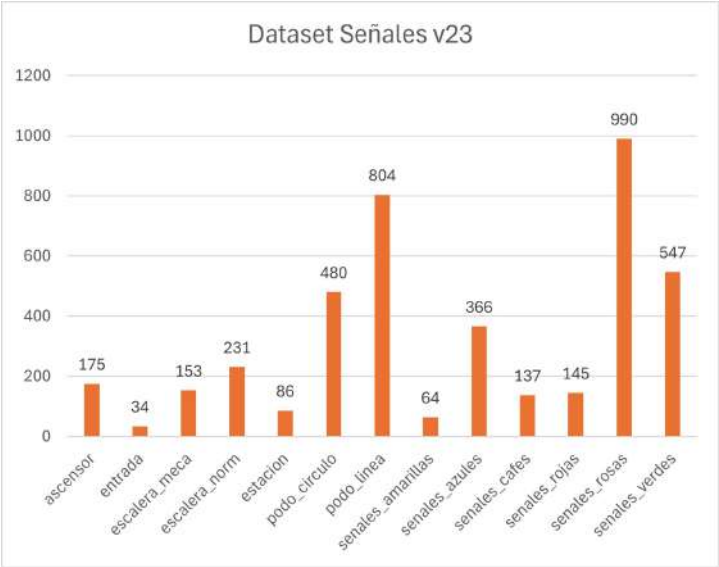


Figura 5.1: *Distribución de instancias etiquetadas para las clases presentes en el dataset de señales-detección.*

descrita en el Capítulo de Requisitos, como por contar con mejor equipamiento y soporte para personas con discapacidad visual respecto a otras líneas de la red, lo que la convierte en el escenario más representativo para el caso de uso principal de AIMES. En consecuencia, se priorizó una mayor densidad de ejemplos de esta señalética específica por sobre una distribución equitativa entre las seis líneas, con el fin de maximizar el desempeño del modelo en el entorno donde el sistema tiene mayor probabilidad de ser utilizado en una primera etapa.

Esta combinación de desbalance intencional (orientado a la Línea 6) y desbalance estructural (asociado a la frecuencia natural de aparición de cada elemento) constituye una consideración relevante para la evaluación del modelo, donde el desempeño se analiza no solo de forma global sino también por clase, con el fin de identificar si las clases con menor representación (como *Entrada* y *Señales Amarillas*) presentan un desempeño de detección inferior al promedio general del modelo.

5.2. Modelo de Detección

Esta sección presenta el análisis comparativo de desempeño entre las tres variantes de arquitectura descritas en el Capítulo de Implementación (YOLOv11n con capa P2, YOLOv11n con bloques RepViT, y YOLOv11n tradicional), evaluadas sobre el conjunto de prueba de la versión final del dataset. El análisis se presenta en el mismo orden en que se definieron las métricas de evaluación: mAP@0.5, F1-Score, Precisión-Recall por clase, y matriz de confusión.

La Figura 5.2 indica cantidades y el resumen de precisión en base al mAP.

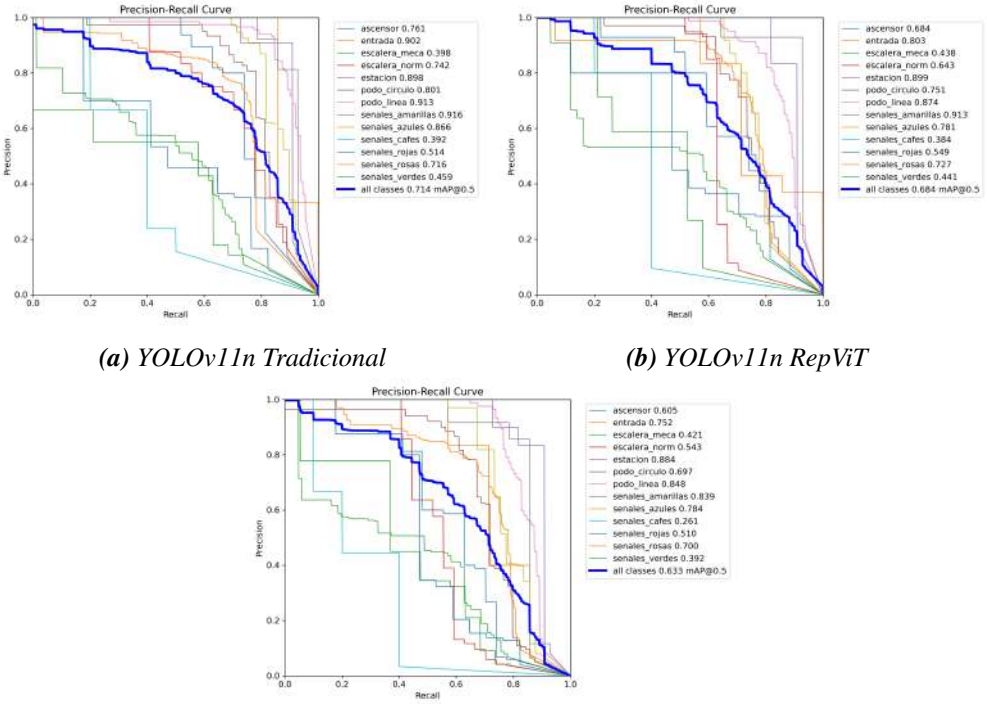
Característica	YOLOv11n + p2	YOLOv11n + RepViT	YOLOv11n Tradicional
Backbone	C2f-ELAN	C2f-ELAN	C2f-ELAN
Configuración	Agrega escala P2 y bloques C3k2 para objetos pequeños.	Backbone tipo RepViT con bloques C3k2 y SPPF.	Por defecto
Precisión mAP@0.5 All Classes	0.56	0.631	0.698
Parámetros estimados	~2.74 millones	> 5 millones	~2.7 millones
Épocas	100	100	100

Figura 5.2: *Comparación de modelos: parámetros, arquitecturas y resultados de mAP obtenidos.*

5.2.1. Evaluación de Métrica: Precisión y Recall

La Figura 5.3 presenta las curvas de Precisión-Recall obtenidas para cada clase, junto con el valor de AP@0.5 individual y el mAP@0.5 global de cada arquitectura.

El análisis por clase revela que el mejor desempeño global de la arquitectura tradicional no es uniforme entre todas las clases: si bien esta supera a las otras dos variantes en 9 de las 13 clases, existen excepciones puntuales: *Escalera Mecánica* obtiene su mejor resultado con RepViT (0,438), y *Señales Rojas* y *Señales Rosas* alcanzan su mejor desempeño con RepViT y P2 respectivamente, aunque con diferencias marginales respecto a la arquitectura tradicional. De forma más relevante, se observa que *Señales Cafés* constituye la clase de peor desempeño de forma consistente en las tres arquitec-



(a) *YOLOv11n Tradicional* ***(b)*** *YOLOv11n RepViT*

(c) *YOLOv11n con P2*

Figura 5.3: *Precision Recall de los Modelo YOLO.*

turas (0,261–0,392), muy por debajo del resto de las señales de color, lo que sugiere una dificultad particular del modelo para distinguir esta clase, independientemente de la arquitectura utilizada. *Escalera Mecánica* y *Señales Verdes* presentan un patrón similar, con valores consistentemente bajos en las tres variantes.

5.2.2. Evaluación de Métrica: mAP@0.5

La Tabla 6 resume el valor de mAP@0.5 global obtenido por cada arquitectura sobre el conjunto de prueba.

La variante tradicional obtiene el mayor valor de mAP@0.5, superando en 8,1 puntos porcentuales a la variante con capa P2 y en 3 puntos porcentuales a la variante con bloques RepViT. Este resultado es relevante considerando que la capa P2 fue incorporada específicamente para mejorar la detección de objetos pequeños, y que los bloques RepViT fueron incorporados para mejorar la robustez frente a variabilidad visual; en

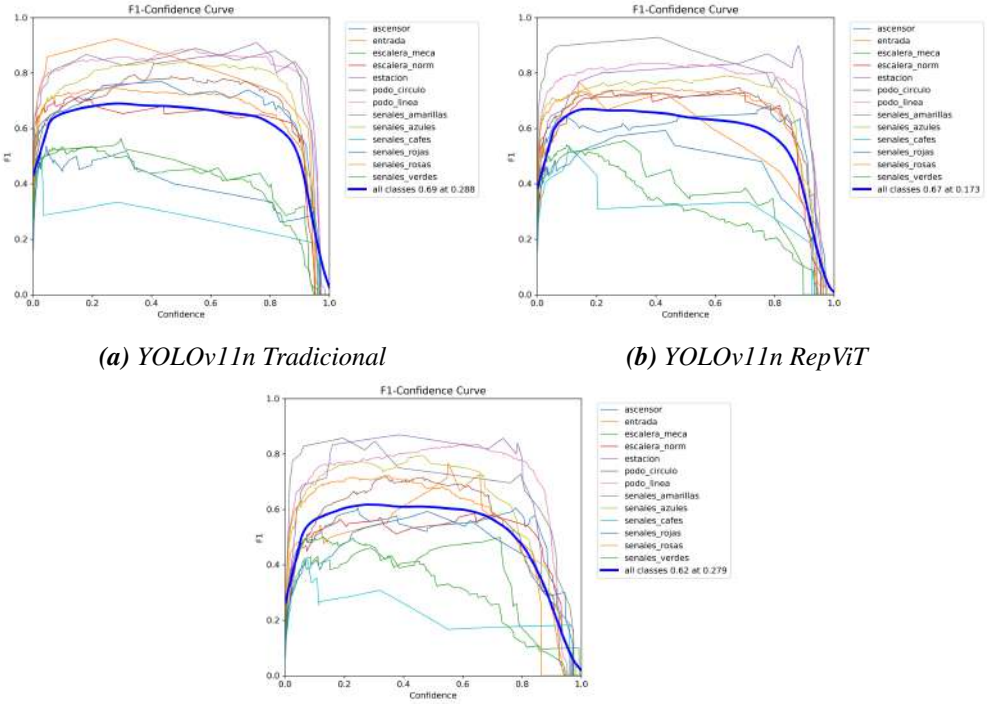
Arquitectura	**mAP@0.5**
YOLOv11n + P2	0,633
YOLOv11n + RepViT	0,684
YOLOv11n Tradicional	0,714

Tabla 6: *Comparación de mAP@0.5 entre arquitecturas*

ambos casos, la complejidad adicional introducida no se tradujo en una mejora de precisión sobre el dataset construido para este trabajo, lo que sugiere que dicha complejidad no logró ser aprovechada adecuadamente, posiblemente debido al tamaño acotado del conjunto de entrenamiento respecto al que normalmente se utiliza para explotar arquitecturas de mayor capacidad.

5.3. Evaluación de Métrica: F1

La Figura 5.4 presenta la curva F1-Confianza para las tres arquitecturas, la cual permite observar tanto el valor máximo de F1 alcanzado como el umbral de confianza en que dicho máximo ocurre.



(a) *YOLOv11n Tradicional* ***(b)*** *YOLOv11n RepViT*

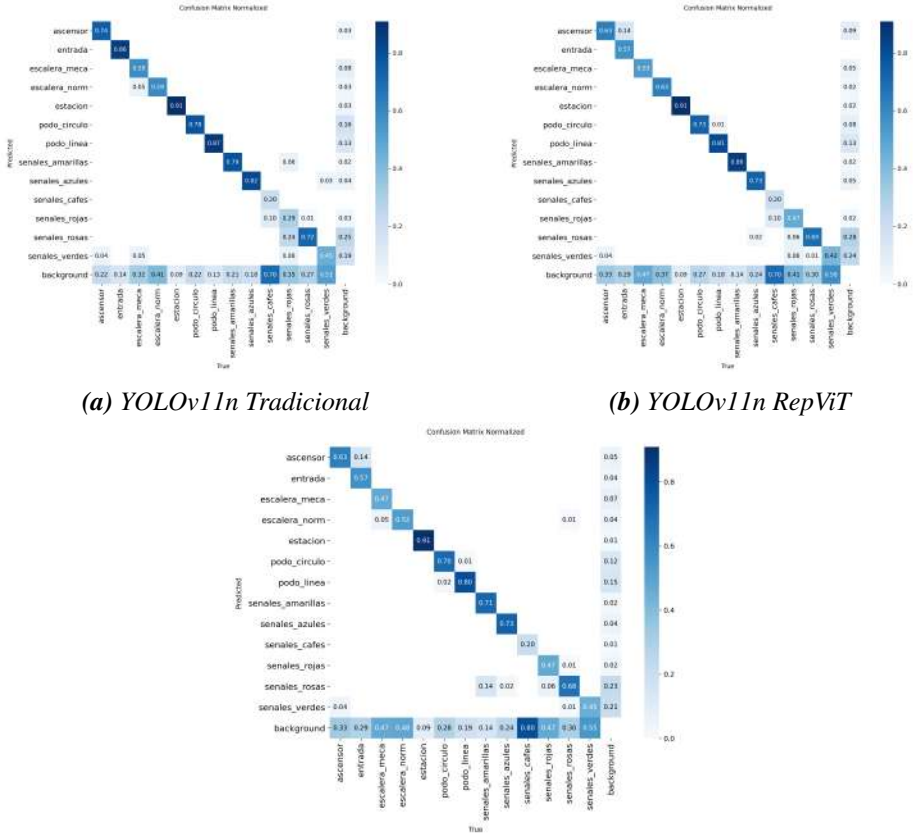
(c) *YOLOv11n con P2*

Figura 5.4: *F1 Score de los Modelo YOLO.*

Consistente con el resultado de mAP@0.5, la arquitectura tradicional alcanza el mayor F1-Score global (0,69). Cabe destacar que este máximo se produce a un umbral de confianza de 0,288, considerablemente más cercano al umbral de 0,70 utilizado como filtro de confianza en la aplicación (Sección de Captura y Procesamiento de Imagen) que el umbral óptimo de la variante RepViT (0,173). Esto sugiere que, en un escenario de uso real donde las detecciones se filtran con un umbral de 0,70, el desempeño relativo de la variante RepViT podría verse afectado en mayor medida que el de la variante tradicional, dado que su punto óptimo de operación se encuentra más alejado del umbral efectivamente utilizado por la aplicación.

5.4. Matriz de confusión

La Figura 5.5 presenta la matriz de confusión normalizada obtenida para cada arquitectura, donde cada columna representa la clase real y cada fila la clase predicha por el modelo, incluyendo una clase adicional de *background* para representar los casos en que un objeto real no fue detectado, o en que se generó una detección sobre una región sin objeto de interés.



(a) *YOLOv11n Tradicional* ***(b)*** *YOLOv11n RepViT*

(c) *YOLOv11n con P2*

Figura 5.5: *Matriz de confusión de los Modelo YOLO.*

La matriz de confusión confirma varios de los hallazgos observados en el análisis de Precisión-Recall. *Señales Cafés* mantiene una tasa de acierto idéntica y notablemente baja (0,20) en las tres arquitecturas, indicando que una parte importante de sus instancias reales son confundidas con otras clases o no detectadas en absoluto. Se observa además una confusión sistemática entre *Escalera Normal* y *Escalera Mecánica*

en las tres variantes, así como cierta confusión entre *Señales Rosas* y las clases *Señales Amarillas* y *Señales Azules*, resultado esperable considerando la similitud visual entre distintos tonos de señalética bajo condiciones de iluminación variable dentro de las estaciones. Finalmente, la columna de *background* muestra que las clases *Podo Línea*, *Señales Rosas* y *Señales Verdes* presentan una proporción considerable de instancias reales no detectadas por el modelo (falsos negativos), un aspecto relevante a considerar más adelante, donde estas clases participan directamente en la validación de dirección mediante OCR.

En conjunto, el análisis de las cuatro métricas evaluadas (mAP@0.5, F1-Score, Precisión-Recall por clase y matriz de confusión) es consistente en señalar a la arquitectura **YOLOv11n tradicional** como la de mejor desempeño global, justificando su selección como modelo base para la conversión y despliegue en la aplicación móvil.

5.5. Valores de benchmark

Esta sección presenta los resultados del benchmark de eficiencia computacional descrito en el Capítulo de Implementación, ejecutado sobre los tres modelos exportados a TensorFlow Lite en el equipo de desarrollo (procesador AMD Ryzen 3400G, inferencia exclusivamente sobre CPU). A diferencia de la sección anterior, centrada en la calidad de detección, aquí se evalúa la viabilidad de cada arquitectura para operar bajo los requisitos no funcionales de tiempo real definidos para el sistema: un mínimo de 20 FPS y una latencia máxima de 200 ms.

La Tabla 7 contiene los valores de evaluación de cada modelo:

Arquitectura	**FPS**	**Latencia (ms)**	**CPU (%)**	**RAM (MB)**
YOLOv11n + P2	12,4	80,6	32	394
YOLOv11n + RepViT	1,0	1008	46	852
YOLOv11n Tradicional	14,6	68,5	34	380

Tabla 7: *Resultados del benchmark de eficiencia computacional*

Los resultados muestran una diferencia radical entre la variante con bloques RepViT y las otras dos arquitecturas: con apenas 1 FPS y una latencia superior a 1 segundo por inferencia, esta variante resulta completamente inviable para un uso en tiempo real, incumpliendo ambos requisitos no funcionales por un margen considerable. Este resultado es coherente con el tamaño del modelo exportado a ONNX observado en el Capítulo de Implementación (203,9 MB, frente a 10,6–17,7 MB de las otras dos variantes), y se explica por el costo computacional propio de un backbone inspirado en Vision Transformers al ejecutarse sobre CPU sin aceleración por hardware especializado.

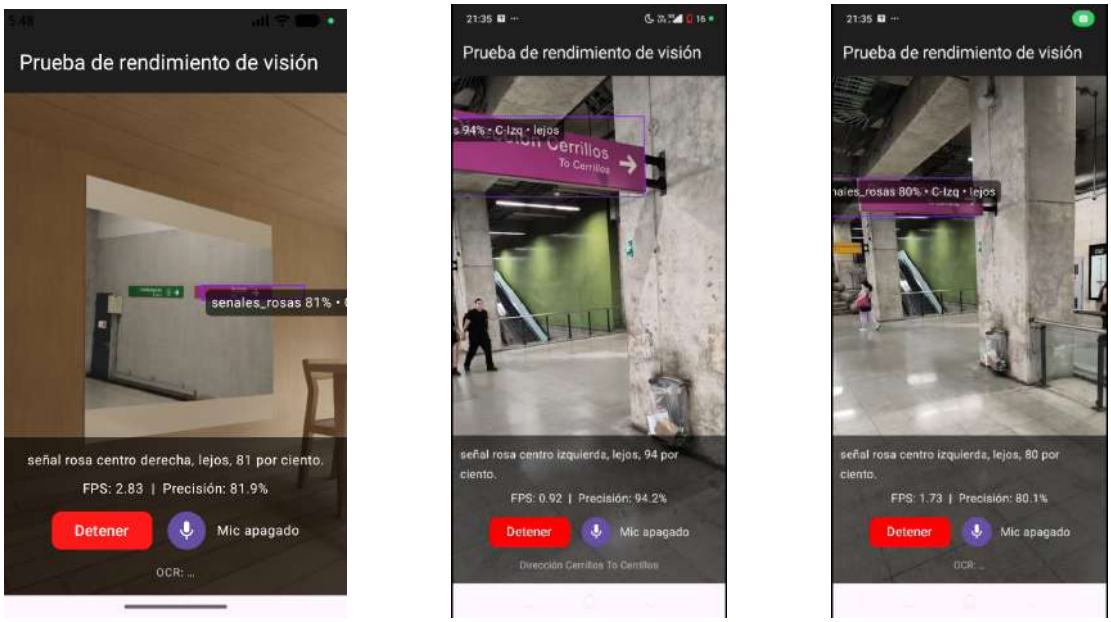
Entre las dos variantes restantes, la arquitectura tradicional presenta un desempeño ligeramente superior a la variante con capa P2, tanto en FPS (14,6 frente a 12,4) como en latencia (68,5 ms frente a 80,6 ms), resultado esperable considerando que la capa P2 introduce una escala adicional de detección (8.500 anclas frente a 8.400, según lo señalado en el Capítulo de Implementación), lo que incrementa levemente la carga computacional sin traducirse, como se vio en la sección anterior, en una mejora de precisión que justifique dicho costo. El uso de CPU se mantiene comparativamente similar entre ambas (32 % y 34 % respectivamente), muy por debajo del 46 % registrado por la variante RepViT.

Es importante destacar que, si bien tanto la arquitectura tradicional como la variante P2 cumplen holgadamente con el umbral de latencia de 200 ms, **ninguna de las tres arquitecturas alcanza el umbral mínimo de 20 FPS** establecido como requisito no funcional, evaluadas en las condiciones de este benchmark. Esto no invalida la selección del modelo, dado que esta evaluación se realizó sobre CPU en un equipo de escritorio y sin cuantización aplicada, condiciones distintas a las del dispositivo móvil final; sin embargo, constituye una advertencia relevante que debe contrastarse con el desempeño observado directamente en la aplicación, presentado en la sección siguiente.

5.6. Aplicación y Uso en Terreno

Con el objetivo de evaluar el funcionamiento de la aplicación AIMES en condiciones de uso, se realizaron pruebas tanto en terreno como en un entorno simulado mediante Android Studio. Estas evaluaciones permitieron observar el comportamiento de AIMES durante la detección de señaléticas del Metro de Santiago, considerando aspectos como la identificación de objetos, el reconocimiento de texto mediante OCR, la estimación de distancia y la generación de retroalimentación al usuario.

La Figura 5.6 reúne las tres capturas analizadas.



(a) *Simulada (Android Studio)* ***(b)*** *Terreno 1* ***(c)*** *Terreno 2*

Figura 5.6: *Capturas de la pantalla de desarrollador durante las pruebas simulada y en terreno.*

5.6.1. Prueba simulada

La prueba simulada se ejecutó sobre el emulador de Android Studio, utilizando su escena virtual de cámara, en la cual se dispuso una fotografía de señalética del Metro. Esto permitió validar el funcionamiento del flujo completo sin necesidad de un despla-

zamiento a terreno. Como se observa en la Figura 5.6(a), el modelo detecta correctamente la señal rosa con una confianza del 81 %, la interfaz la describe como ubicada al centro-derecha y a larga distancia, y la misma descripción se entrega por voz. En esta prueba el campo OCR permanece vacío, dado el reducido tamaño con que la señal aparece dentro de la escena.

5.6.2. Pruebas en terreno

Las pruebas en terreno se realizaron en una estación de la Línea 6 del Metro de Santiago, utilizando un teléfono Android. En la Figura 5.6(b), el sistema detecta una señalética con una confianza del 94 % y reconoce mediante OCR el texto completo "Dirección Cerrillos To Cerrillos", permitiendo contrastar la información con la estación esperada según el trayecto declarado.

Por otro lado, la Figura 5.6(c), muestra que el sistema detecta correctamente una señal con una confianza del 80 %, pero no logra reconocer su contenido mediante OCR debido a la distancia de captura y la baja resolución del texto.

Estas pruebas permiten verificar cualitativamente la integración entre el módulo de detección y el reconocimiento de texto. Cuando la señalética se encuentra suficientemente cercana y su contenido presenta resolución adecuada, el sistema logra extraer mediante OCR información asociada al sentido de viaje. En cambio, a mayores distancias el modelo puede detectar correctamente la presencia de la señal mientras el texto deja de ser legible para el reconocedor.

Por lo tanto, las pruebas confirman el funcionamiento del flujo integrado propuesto, pero también evidencian que el desempeño del OCR depende fuertemente del tamaño y la calidad de la región de texto disponible. Esta observación constituye una limitación de la implementación actual y motiva las mejoras de preprocesamiento planteadas como trabajo futuro.

Finalmente, las pruebas realizadas corresponden a una validación técnica del sistema y no a una evaluación de usabilidad con personas con discapacidad visual. En

consecuencia, los resultados permiten evaluar el comportamiento de los módulos implementados, pero no permiten concluir todavía sobre la efectividad, seguridad o experiencia de uso de AIMES por parte de su población objetivo. Una validación posterior con usuarios constituye una etapa necesaria antes de plantear su utilización como sistema de asistencia en condiciones reales.

6. Conclusiones

El desarrollo de AIMES permitió abordar la problemática de orientación que enfrentan las personas con discapacidad visual dentro de entornos complejos como el Metro de Santiago, mediante una solución tecnológica capaz de interpretar información del entorno y transformarla en indicaciones accesibles para el usuario. El sistema fue concebido como un complemento a las herramientas de movilidad ya utilizadas por estas personas (principalmente el bastón) y no como un reemplazo, enfoque que surgió directamente del levantamiento de requisitos realizado con usuarios y organizaciones especializadas. A partir de los cinco objetivos específicos planteados en esta memoria, es posible evaluar el grado de cumplimiento alcanzado en cada etapa del trabajo.

6.1. Conclusión de Trabajo

Respecto a la **construcción y evaluación del dataset**, se logró diseñar un conjunto de datos especializado para el entorno del Metro de Santiago, ante la inexistencia de repositorios públicos equivalentes. El proceso, documentado a través de ocho versiones sucesivas (v16 a v23), evidenció que la construcción de un dataset corresponde a un proceso iterativo de ajuste (entre volumen de datos, técnicas de aumento y calidad del etiquetado) más que a una decisión única tomada al inicio. La versión final, con 13 clases que cubren tanto infraestructura fija como señalética direccional, respondió tanto a la estructura de Metro como a las necesidades identificadas en terreno, aunque con una distribución de clases desbalanceada, en parte intencional (priorizando la Línea 6) y en parte estructural.

En cuanto al **entrenamiento y evaluación de modelos de detección**, se compararon tres variantes de arquitectura basadas en YOLOv11n (tradicional, con capa P2, y con bloques RepViT), seleccionando la variante tradicional por obtener el mejor desempeño global (mAP@0.5 de 0,714) junto con la mejor eficiencia computacional. Este resultado es significativo porque contradice la expectativa inicial de que una ma-

yor complejidad arquitectónica mejoraría el desempeño: ni la capa P2 ni los bloques RepViT lograron traducir su complejidad adicional en mejores resultados sobre este dataset, y la variante RepViT resultó además inviable para tiempo real, con apenas 1 FPS en el benchmark de eficiencia.

Sobre la **optimización para dispositivos móviles**, el modelo fue exportado exitosamente a TensorFlow Lite, pero no fue posible aplicar cuantización: las variantes cuantizadas generaron fallos irrecuperables al inicializar el intérprete, forzando a mantener el modelo en precisión completa (float32). Esta limitación, no resuelta dentro de los plazos del proyecto, se refleja en el desempeño observado: ni en el benchmark de escritorio (14,6 FPS en el mejor caso) ni en las pruebas en terreno (entre 0,92 y 2,83 FPS) se alcanzó el umbral de 20 FPS definido como requisito no funcional. Esta brecha constituye la limitación más relevante del sistema en su estado actual.

En el **desarrollo de la aplicación móvil**, se implementó de forma nativa en Kotlin un sistema capaz de capturar video en tiempo real, ejecutar el modelo de detección localmente e interpretar sus resultados en términos de posición y distancia. Esta decisión de migrar desde un prototipo inicial en Python/Kivy/Buildozer hacia un desarrollo nativo evidenció que no basta con validar el funcionamiento individual de cada tecnología, sino también su capacidad real de integración y estabilidad dentro de un sistema con procesamiento continuo. Dentro de esta aplicación, el componente de reconocimiento óptico de caracteres constituye el aporte individual más profundo de este trabajo: mientras el modelo de detección permite reconocer la presencia de una señalética, el módulo OCR permite contrastar su contenido textual con la estación de destino declarada, diferenciando entre detectar un letrero y validar si ese letrero corresponde efectivamente a la ruta solicitada. Las pruebas en terreno confirmaron su funcionamiento en condiciones reales, aunque también su principal limitante: la legibilidad del texto depende directamente de la distancia de captura, y la estimación de distancia basada en la altura de la caja delimitadora no resulta adecuada para señaléticas alargadas horizontalmente.

Finalmente, en cuanto a la **validación de la solución**, las pruebas realizadas (simu-

ladas y en terreno en la Línea 6 del Metro de Santiago, específicamente en la estación Franklin) permitieron comprobar el funcionamiento conjunto de detección, estimación espacial y reconocimiento de texto en un escenario representativo. Estas mismas pruebas, sin embargo, delimitan con claridad el alcance actual de AIMES: corresponde a un prototipo funcional de asistencia y orientación, no a un sistema completo de navegación autónoma. El módulo de alertas de emergencia y la detección de servicios y salidas específicas, identificados como relevantes durante el levantamiento de requisitos, no alcanzaron a implementarse dentro de este trabajo.

El grado de cumplimiento de los objetivos planteados fue diferente entre las distintas etapas del proyecto. La construcción del dataset, el entrenamiento y comparación de modelos y el desarrollo de la aplicación móvil fueron completados de acuerdo con el alcance definido. En cambio, la optimización para dispositivos móviles se alcanzó de forma parcial: el modelo pudo convertirse e integrarse mediante TensorFlow Lite, pero la cuantización no pudo ser incorporada de manera funcional y el rendimiento obtenido se mantuvo por debajo del umbral de 20 FPS establecido como requisito.

De manera similar, la validación permitió comprobar técnicamente el funcionamiento integrado del sistema mediante pruebas simuladas y en terreno, pero no incluyó una evaluación cuantitativa de usabilidad ni pruebas con personas con discapacidad visual. Por esta razón, no es posible afirmar todavía que la solución haya sido validada desde la perspectiva de su población objetivo.

En consecuencia, el objetivo general puede considerarse alcanzado en cuanto al diseño de implementación de un prototipo móvil capaz de detectar e interpretar elementos del entorno, pero solo parcialmente en lo referido a su operación en tiempo real bajo el criterio de rendimiento definido en esta memoria.

6.2. Aportes

Como principales aportes de este trabajo se identifican la construcción de un dataset especializado en infraestructura y señalética del Metro de Santiago; la comparación ex-

perimental de diferentes configuraciones de YOLOv11n; la integración de un modelo de detección dentro de una aplicación Android con procesamiento local; y la combinación de detección visual, OCR y retroalimentación auditiva dentro de un mismo flujo de asistencia.

Un aporte particularmente relevante corresponde al uso complementario de detección y OCR. La detección permite delimitar primero una región correspondiente a señalética, mientras el reconocimiento de texto permite posteriormente determinar si su contenido es coherente con el destino solicitado por el usuario. Esta separación reduce la necesidad de procesar indiscriminadamente todo el texto presente en la escena y aproxima el sistema a un comportamiento contextual orientado a la navegación.

6.3. Limitaciones

Las principales limitaciones corresponden al rendimiento inferior al umbral de tiempo real definido, la imposibilidad de aplicar cuantización al modelo final, la dependencia del OCR respecto de la distancia y resolución del texto, el desbalance entre clases del dataset y el carácter aproximado de la estimación de distancia basada únicamente en el tamaño de las cajas delimitadoras.

Asimismo, las pruebas realizadas no incluyeron participantes con discapacidad visual, por lo que no se evaluaron aspectos fundamentales como comprensibilidad de las instrucciones, carga auditiva, seguridad durante el desplazamiento o utilidad percibida. En consecuencia, AIMES debe entenderse en su estado actual como un prototipo tecnológico validado a nivel funcional y no como una solución lista para utilización autónoma por usuarios finales.

7. Trabajo Futuro

A partir del desarrollo y los resultados obtenidos con AIMES, se identifican distintas oportunidades de mejora orientadas principalmente al módulo de visión por computador y a la entrega oportuna de información al usuario. Estas mejoras buscan aumentar la precisión, velocidad y robustez del sistema para su utilización en escenarios reales dentro del Metro de Santiago.

7.1. Optimización del modelo de detección

Como trabajo futuro, se propone continuar con la optimización del modelo de detección utilizado por AIMES, buscando mejorar tanto su precisión como su velocidad de inferencia. Para ello, se pueden evaluar nuevas arquitecturas o versiones más actualizadas de modelos compatibles con dispositivos móviles, además de realizar ajustes en los parámetros de entrenamiento y técnicas de optimización.

Asimismo, resulta relevante evaluar el comportamiento del modelo bajo diferentes condiciones de iluminación, movimiento, distancia, ángulo de captura y oclusión. El objetivo es reducir el tiempo necesario para procesar cada imagen y aumentar la frecuencia de detecciones, manteniendo un equilibrio entre precisión y rendimiento en tiempo real.

7.2. Ampliación y mejora del dataset

Se propone incorporar una mayor cantidad y diversidad de imágenes obtenidas en estaciones del Metro de Santiago, incluyendo nuevos elementos relevantes para la orientación y autonomía del usuario, tales como puntos de recarga de tarjetas Bip!, cajeros automáticos, locales comerciales, accesos específicos y señalización asociada a salidas y servicios.

Además, el dataset puede considerar imágenes capturadas en diferentes estaciones,

horarios, condiciones de iluminación, distancias y ángulos, así como escenarios con personas u objetos que generen oclusiones parciales. Una mayor diversidad de datos permitiría mejorar la capacidad de generalización del modelo y disminuir errores frente a situaciones que no fueron representadas suficientemente durante el entrenamiento inicial.

7.3. Mejora del reconocimiento mediante OCR

Respecto al módulo OCR, se propone mejorar el procesamiento de las regiones de imagen que contienen información textual antes de realizar el reconocimiento. Para ello, pueden incorporarse técnicas de preprocesamiento como corrección de perspectiva, ajuste de contraste, reducción de ruido y recorte de regiones de interés.

Adicionalmente, el texto reconocido puede ser validado utilizando información contextual, como la estación actual y el destino seleccionado por el usuario. De esta manera, AIMES podría determinar qué información de la señalética resulta relevante para el desplazamiento y descartar aquella que no contribuya directamente a la orientación.

7.4. Comunicación de instrucciones al usuario

Como extensión del módulo OCR, se propone optimizar la transformación de la información reconocida en instrucciones auditivas breves, claras y oportunas. El sistema deberá priorizar la información relevante para la ruta del usuario, evitando comunicar de manera literal todo el texto identificado en el entorno. Actualmente, este manejo se realiza de manera parcial, priorizando la información asociada al objeto de interés solicitado por el usuario y comunicando su orientación. Como trabajo futuro, se plantea ampliar esta capacidad mediante una selección más contextual de la información, permitiendo comunicar elementos adicionales que puedan resultar relevantes durante el desplazamiento, sin saturar al usuario con información innecesaria proveniente del

entorno.

Esto permitiría mejorar la autonomía del usuario con información relevante sobre el entorno, reducir la cantidad de información entregada mediante síntesis de voz y facilitar su comprensión durante el desplazamiento. Además, se plantea mejorar la coordinación entre los módulos de OCR, interpretación y TTS, de manera que exista una menor latencia entre la detección de una señalética y la entrega de la instrucción correspondiente.

7.5. Optimización del flujo de procesamiento en tiempo real

Finalmente, se propone optimizar el flujo completo de procesamiento de AIMES, considerando las etapas de captura de imagen, detección de objetos, reconocimiento de texto, interpretación de resultados y generación de instrucciones auditivas.

La reducción de la latencia en este ciclo permitiría que las indicaciones entregadas representen de mejor manera el estado actual del entorno mientras el usuario se encuentra en movimiento. Por lo tanto, futuros desarrollos deberán considerar no solo la precisión individual de cada módulo, sino también el tiempo total requerido desde la captura de información hasta su comunicación mediante voz.

Referencias

[1] Ministerio de Desarrollo Social y Familia. *Resultados Encuesta Nacional de Discapacidad y Dependencia 2022: Población adulta*. Inf. téc. Accessed: Oct. 3, 2026. Gobierno de Chile, 2022. URL: https://observatorio.ministeriodesarrollosocial.gob.cl/storage/docs/endide/230128_Resultados_ENDIDE_2022(Poblacion-adulta).pdf.

[2] World Health Organization. *Ceguera y discapacidad visual*. Accessed: Oct. 3, 2026. Ago. de 2023. URL: https://www.who.int/es/news-room/fact-sheets/detail/blindness-and-visual-impairment.

[3] Directorio de Transporte Público Metropolitano. *Informe de Gestión 2025*. Inf. téc. Accessed: Oct. 2, 2026. Directorio de Transporte Público Metropolitano, 2025. URL: https://www.dtpm.cl/descargas/memoria/Informe%20de%20Gestion%20DTPM2025.pdf.

[4] Metro de Santiago. *Metro Accesible*. Accessed: Oct. 2, 2026. URL: https://www.metro.cl/guia-del-usuario/metro-accesible.

[5] Francia Quezada. *Reportaje revela que ambulantes han dañado cerca del 30 % de los ascensores del Metro de Santiago*. BioBioChile. Published: Jun. 5, 2023. Accessed: Oct. 3, 2026. Jun. de 2023. URL: https://www.biobiochile.cl/noticias/nacional/region-metropolitana/2023/06/05/reportaje-revela-que-ambulantes-han-danado-cerca-del-30-de-los-ascensores-del-metro-de-santiago.shtml.

[6] Tim Brown. *Change by Design: How Design Thinking Creates New Alternatives for Business and Society*. Boston, MA, USA: Harvard Business Press, 2009.

[7] Fundación Luz. *Estudio sobre Personas con Discapacidad Visual en Chile – Fundación Luz / ENDIDE 2022*. Published: Oct. 8, 2025. Accessed: Oct. 3, 2026.

2025. URL: https://fundacionluz.cl/area/estudio-sobre-personas-con-discapacidad-visual-en-chile-fundacion-luz-endide-2022/.

[8] Biblioteca del Congreso Nacional de Chile. *Ley N.º 20.422: Establece normas sobre igualdad de oportunidades e inclusión social de personas con discapacidad*. Updated text accessed: Oct. 3, 2026. 2010. URL: https://www.bcn.cl/leychile/navegar?idNorma=1010903.

[9] Servicio Nacional de la Discapacidad. *Metro inaugura accesos para personas con discapacidad en estación Universidad de Chile*. Published: Aug. 17, 2009. Accessed: Oct. 3, 2026. Ago. de 2009. URL: https://www.senadis.gob.cl/sala_prensa/d/noticias/142/metro-inaugura-accesos-para-personas-con-discapacidad-en-estacion-universidad-de-chile.

[10] Servicio Nacional de la Discapacidad. *Metro avanza en su plan de accesibilidad y habilita tres ascensores en estación El Llano de Línea 2*. Published: Aug. 23, 2018. Accessed: Oct. 3, 2026. Ago. de 2018. URL: https://www.senadis.gob.cl/region/metropolitana/d/portadas/7445/metro-avanza-en-su-plan-de-accesibilidad-y-habilita-tres-ascensores-en-estacion-el-llano-de-linea-2.

[11] Metro de Santiago. *Lazarillo, la aplicación que ayuda a las personas ciegas se expande en la red de Metro y alcanza las 13 estaciones*. Accessed: Oct. 3, 2026. Nov. de 2023. URL: https://www.metro.cl/noticias/lazarillo-la-aplicacion-que-ayuda-a-las-personas-ciegas-se-expande-en-la-red-de-metro-y-alcanza-las-13-estaciones.

[12] Perkins School for the Blind. *Lazarillo: A Free Accessible GPS App for the Blind and Visually Impaired*. Accessed: Oct. 3, 2026. Jul. de 2024. URL: https://www.perkins.org/resource/lazarillo-free-accessible-gps-app-blind-and-visually-impaired/.

[13] Perkins School for the Blind. *Microsoft Seeing AI and Low Vision Review*. Accessed: Oct. 3, 2026. Oct. de 2022. URL: https://www.perkins.org/resource/microsoft-seeing-ai-and-low-vision-review/.

[14] Erum Salam. *I Tried Be My Eyes, the Popular App That Pairs Blind People with Helpers*. The Guardian. Accessed: Oct. 3, 2026. Jul. de 2019. URL: https://www.theguardian.com/lifeandstyle/2019/jul/12/be-my-eyes-app-blind-people-helpers.

[15] Perkins School for the Blind. *Using Aira with Low Vision*. Accessed: Oct. 3, 2026. Jul. de 2023. URL: https://www.perkins.org/resource/aira-low-vision-review/.

[16] Perkins School for the Blind. *Visual Assistance Apps: Post Roundup*. Accessed: Oct. 3, 2026. Ago. de 2023. URL: https://www.perkins.org/resource/visual-assistance-apps-post-roundup/.

[17] Ghatan Fauzi Nugraha, Aradea, Rianto y Rina Mardiati. «Object Detection for the Visually Impaired: A Systematic Literature Review». En: *Proc. 10th Int. Conf. Wireless Telematics (ICWT)*. 2024, págs. 1-16. DOI: 10.1109/ICWT62080.2024.10674727.

[18] Pal Patel, Shreyansh Pampaniya, Ananya Ghosh, Ritu Raj, Deepa Karuppaih y Saravanakumar Kandasamy. «Enhancing Accessibility Through Machine Learning: A Review on Visual and Hearing Impairment Technologies». En: *IEEE Access* 13 (2025), págs. 33286-33307. DOI: 10.1109/ACCESS.2025.3539081.

[19] Joseph Redmon, Santosh Divvala, Ross Girshick y Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. arXiv: 1506.02640 [cs.CV]. URL: https://arxiv.org/abs/1506.02640.

[20] SO Development. *Comparing YOLOv11 and YOLOv12: A Deep Dive into the Next-Generation Object Detection Models*. Accessed: Oct. 3, 2026. Feb. de 2025. URL: https://so-development.org/comparing-yolov11-and-yolov12-

a-deep-dive-into-the-next-generation-object-detection-models/.

[21] Rahima Khanam y Muhammad Hussain. *A Review of YOLOv12: Attention-Based Enhancements vs. Previous Versions*. 2025. arXiv: 2504.11995 [cs.CV]. URL: https://arxiv.org/abs/2504.11995.

[22] Ao Wang, Hui Chen, Zijia Lin, Jungong Han y Guiguang Ding. «RepViT: Revisiting Mobile CNN From ViT Perspective». En: *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*. 2024, págs. 15909-15920. DOI: 10.1109/CVPR52733.2024.01506.

[23] Floriana Ciaglia, Francesco Saverio Zuppichini, Paul Guerrie, Mark McQuade y Jacob Solawetz. *Roboflow 100: A Rich, Multi-Domain Object Detection Benchmark*. 2022. arXiv: 2211.13523 [cs.CV]. URL: https://arxiv.org/abs/2211.13523.

[24] Google. *LiteRT: High-Performance On-Device Machine Learning Framework*. Google AI Edge. Accessed: Oct. 2, 2026. URL: https://developers.google.com/edge/litert.

[25] JetBrains. *Kotlin Documentation*. Accessed: Sep. 29, 2026. 2026. URL: https://kotlinlang.org/docs/.

[26] World Wide Web Consortium. *Web Content Accessibility Guidelines (WCAG) 2.1*. W3C Recommendation, Jun. 5, 2018. Accessed: Oct. 3, 2026. 2018. URL: https://www.w3.org/TR/WCAG21/.

[27] Google. *Descripción general de CameraX*. Android Developers. Accessed: Sep. 29, 2026. URL: https://developer.android.com/media/camera/camerax?hl=es-419.

[28] Google. *Reconoce texto en imágenes con ML Kit en Android*. Google for Developers. Accessed: Sep. 29, 2026. URL: https://developers.google.com/ml-kit/vision/text-recognition/v2/android?hl=es-419.

[29] Google. *TextToSpeech*. Android Developers. Accessed: Sep. 29, 2026. URL: https://developer.android.com/reference/android/speech/tts/TextToSpeech.

[30] Google. *SpeechRecognizer*. Android Developers. Accessed: Sep. 29, 2026. URL: https://developer.android.com/reference/android/speech/SpeechRecognizer.

[31] Google. *Jetpack Compose*. Android Developers. Accessed: Sep. 29, 2026. URL: https://developer.android.com/develop/ui/compose.

[32] Ratnesh Litoriya, Kailash Chandra Bandhu, Sanket Gupta, Ishika Rajawat, Hany Jagwani y Chirayu Yadav. «Implementing Visual Assistant Using YOLO and SSD for Visually-Impaired Persons». En: *Journal of Automation, Mobile Robotics and Intelligent Systems* 17.4 (2024), págs. 79-87. DOI: 10.14313/JAMRIS/4-2023/33.

[33] Vinish Pujari, Karthik Madnal y Divya Premchandran. «Mobile App for Enhancing Accessibility Among the Visually Impaired». En: *Proc. 2nd DMIHER Int. Conf. Artif. Intell. Healthcare, Educ. Ind. (IDICAIEI)*. 2024, págs. 1-6. DOI: 10.1109/IDICAIEI61867.2024.10842749.

[34] Wei Shi, Rui Shan y Yoshihiro Okada. «A Navigation System for Visual Impaired People Based on Object Detection». En: *Proc. 12th Int. Congr. Adv. Appl. Informatics (IIAI-AAI)*. 2022, págs. 354-358. DOI: 10.1109/IIAIAAI55812.2022.00078.

[35] Xinrui Yu y Jafar Saniie. «Visual Impairment Spatial Awareness System for Indoor Navigation and Daily Activities». En: *Journal of Imaging* 11.1 (2025), pág. 9. DOI: 10.3390/jimaging11010009.

[36] Deqiang He, Zhou Jiang, Jiyong Chen, Jianren Liu, Jian Miao y Abid Shah. «Classification of Metro Facilities with Deep Neural Networks». En: *Journal of Advanced Transportation* 2019 (2019). Article ID 6782803, págs. 1-8. DOI: 10.1155/2019/6782803.

[37] Google. *Android Studio*. Android Developers. Accessed: Oct. 3, 2026. URL: https://developer.android.com/studio.

[38] Instituto Nacional de Estadísticas. *Censo 2024 – Primeros resultados*. Accessed: Oct. 3, 2026. 2024. URL: https://www.ine.gob.cl.

[39] Karl T. Ulrich y Steven D. Eppinger. *Product Design and Development*. 5th. New York, NY, USA: McGraw-Hill/Irwin, 2012.

[40] Kivy Authors. *Kivy Documentation*. Accessed: Oct. 3, 2026. 2011. URL: https://kivy.org/doc/stable/.

[41] Kivy. *Plyer: A Platform-Independent Python Wrapper for Platform-Dependent APIs*. GitHub. Accessed: Oct. 3, 2026. URL: https://github.com/kivy/plyer.

[42] Olga Yakovleva. *RHVoice*. GitHub. Accessed: Oct. 3, 2026. 2022. URL: https://github.com/RHVoice/RHVoice.

[43] Alpha Cephei. *Vosk Speech Recognition Toolkit*. Accessed: Oct. 3, 2026. 2022. URL: https://alphacephei.com/vosk/.

[44] AIMES. *Senales Computer Vision Model*. Roboflow Universe. Accessed: Oct. 3, 2026. 2025. URL: https://universe.roboflow.com/aimes/senales-cnijl.

[45] AIMES. *Senales-segmentation Computer Vision Dataset*. Roboflow Universe. Accessed: Oct. 3, 2026. 2025. URL: https://universe.roboflow.com/aimes/senales-segmentation-zvzpo.

[46] Glenn Jocher y Jing Qiu. *Ultralytics YOLO11*. Version 11.0.0. Accessed: Oct. 3, 2026. 2024. URL: https://github.com/ultralytics/ultralytics.

[47] Mario Almeida, Stefanos Laskaridis, Abhinav Mehrotra, Lukasz Dudziak, Ilias Leontiadis y Nicholas D. Lane. *Smart at What Cost? Characterising Mobile Deep Neural Networks in the Wild*. 2021. arXiv: 2109.13963 [cs.LG]. URL: https://arxiv.org/abs/2109.13963.

[48] TensorFlow. *Introducción a los tensores*. TensorFlow Core. Accessed: Sep. 23, 2026. 2024. URL: https://www.tensorflow.org/guide/tensor.

Anexos

.1. Pauta de entrevista en terreno a Don Iván

La siguiente pauta corresponde al conjunto de preguntas utilizadas como guía durante la entrevista en terreno.

1. ¿Cómo se moviliza habitualmente dentro del Metro y qué herramientas utiliza para orientarse?
   Me movilizo de forma independiente dentro del Metro y utilizo principalmente mi bastón para orientarme por las estaciones.

2. ¿Qué dificultades encuentra al interactuar con elementos o dispositivos presentes dentro de las estaciones?
   Una de las principales dificultades que tengo es interactuar con las pantallas, especialmente con las máquinas para recarga. Muchas veces no tengo idea de qué parte de la pantalla estoy tocando, por lo que se me dificulta realizar algunas cosas.

3. ¿De qué manera una aplicación o sistema tecnológico podría ayudarlo frente a estas dificultades?
   Creo que una aplicación o sistema podría ayudarme indicando dónde debo tocar en la pantalla o leyendo la información que aparece en ella, así la puedo usar sin pedirle ayuda a nadie.

4. Además de las máquinas de recarga, ¿qué otro tipo de información considera útil conocer dentro de una estación
   Me sería útil conocer qué servicios tengo cerca dentro de la estación, por ejemplo, cajeros automáticos o tiendas como Oxxo o Pronto. A veces necesito acceder a estos lugares, pero no tengo referencias claras de dónde están.

5. ¿Existen dificultades para determinar su ubicación dentro del andén después de bajar del tren?

   Si muchas veces, cuando bajo del tren, no sé realmente si estoy al principio, al medio o al final del andén. Eso dificulta la orientación y saber hacia dónde debo continuar.

6. ¿Qué información considera que podría ayudarlo a orientarse mejor al momento de salir de una estación?

   Me ayudaría que la aplicación pudiera indicarme cuáles son las salidas de la estación y a qué dirección corresponde cada una. Así me da más libertad de elegir a qué salida tengo que tomar o hacia donde me debo dirigir.

7. Pensando en una solución tecnológica para apoyar el desplazamiento, ¿qué tipo de dispositivo considera que podría ser útil?

   Creo que podría ser útil una cámara externa, acompañada de un altavoz. Ya que al usar el teléfono constantemente igual corro el riesgo a que me puedan robar, pero es más cómodo igual.

8. ¿Hay alguna otra situación dentro del Metro que considere importante abordar mediante una solución tecnológica de asistencia?

   Sí, sería útil que la aplicación avisara cuando hay cambios en la estación, como ascensores malos, escaleras cerradas o mucha gente en el camino.