



UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY



FACULTAD DE
INGENIERÍA

Extensión de Robotito mediante visión con un dispositivo móvil

Informe de Proyecto de Grado presentado por: Ramiro Moreira.

16 de julio de 2025

En cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores:

Jorge Visca, Ewelina Bakala, Mercedes Marzoa Tanco

Agradecimientos

A Jorge, Ewelina y Mercedes, por la infinita paciencia en el largo proceso que llevo el proyecto, por las muchas llamadas, montones de mails, correcciones, buenas ideas y bajadas a tierra oportunas. A Joanna por discutir párrafo a párrafo para mejorar mi redacción y por prestarse para probar a Robotito.

Resumen

Robotito es un robot educativo diseñado para trabajar con niños, quienes a partir de modificar su entorno son capaces de cambiar el comportamiento del robot. Antes del presente proyecto, Robotito contaba con dos tipos de sensores (sensor de color y distancia), además de sus ruedas omnidireccionales y anillo de luces leds como actuadores. Este proyecto extiende la interacción de Robotito mediante la incorporación de un dispositivo móvil, de forma de utilizar la cámara para proporcionar reconocimiento de imágenes, y su pantalla como actuador. Para la implementación del reconocimiento de imágenes se utilizaron redes neuronales en el dispositivo, implementadas mediante modelos de Tensorflow lite. En cuanto a la lógica que gobierna el comportamiento del robot, se utilizaron máquinas de estados definidas en un repositorio online. Se experimentó el funcionamiento de los modelos de reconocimiento de imágenes utilizados y del robot en general, obteniendo tiempos aceptables de ejecución de los mismos (menores a 500 ms) con una precisión de 97.8 %. Además se efectuó un caso de uso en el cual el robot navega por un laberinto a medida que detecta figuras geométricas.

Índice

1. Introducción	8
2. Revisión de antecedentes	11
2.1. Robótica educativa	11
2.2. Robots educativos con visión por celular	13
2.3. Robotito	17
3. Diseño de solución	22
3.1. Funcionalidades propuestas	24
3.1.1. Definir un comportamiento	24
3.1.2. Seleccionar un comportamiento	26
3.1.3. Control remoto	26
3.1.4. Detección de marcas	26
3.1.5. Etiquetado de imágenes (general)	26
3.1.6. Etiquetado de imágenes (figuras geométricas)	27
3.1.7. Reproducir sonidos	27
3.1.8. Desplegar imágenes/gifs en pantalla	27
3.2. Consideraciones generales	28
3.3. Arquitectura general	28
4. Implementación	32
4.0.1. Plataforma	32
4.0.2. Framework de desarrollo	33
4.0.3. Modelos de reconocimiento de imágenes	34
4.0.4. Firebase	35
4.0.5. MLKit	35
4.0.6. Google Cloud	36
4.1. Arquitectura implementada	37
4.1.1. Componentes	38
4.1.2. Comportamientos	42
4.1.3. Reconocimiento de imágenes, marcas	47
4.1.4. Reconocimiento de imágenes, general	48
4.1.5. Reconocimiento de Figuras geométricas	48
5. Experimentación	54
5.1. Pruebas de funcionalidades	54
5.2. Mediciones de funcionamiento	58
6. Conclusiones y trabajo a futuro	63

1. Introducción

El presente proyecto “Extensión de Robotito mediante visión con un dispositivo móvil” tiene como objetivo expandir las posibilidades que ofrece el robot educativo Robotito (Grupo MINA, 2019). Mediante la incorporación de un dispositivo móvil y el desarrollo de una aplicación, se busca mejorar sus capacidades de interacción con su entorno, utilizando como sensor la cámara para realizar reconocimiento de imágenes, además de la pantalla y el sonido del celular como actuadores.

La robótica educativa, entendida como “el conjunto de actividades pedagógicas que apoyan y fortalecen áreas específicas del conocimiento a través de la concepción, creación, ensamble y puesta a punto de robots” [1], está estrechamente ligada con el aprender haciendo (del inglés “learning by doing” [2]), método que se basa en que se adquiera el conocimiento mediante la acción y la puesta en práctica. Entre sus beneficios más destacados se encuentra el ayudar a desarrollar el pensamiento computacional, el trabajo en equipo y la capacidad de resolución de problemas [3]. Algunas cualidades deseables en un robot educativo son que ofrezca diversas y variadas formas de interacción con sus usuarios, que tenga flexibilidad para ser utilizado en la enseñanza de distintas áreas, que sea atractivo para los usuarios y que sea simple de usar.

Robotito [4] busca tener estas cualidades y se presenta como una herramienta pedagógica útil que permite a niños y niñas controlar el comportamiento de un robot manipulando su entorno. El robot (figura 1), de forma circular, cuenta actualmente con varios sensores y actuadores: un sensor del color del suelo, sensores de distancia que le permiten detectar los objetos próximos, utiliza motores para su movimiento y además dispone de luces leds en la parte superior.



Figura 1: Robotito en varios colores (al frente) y primeros prototipos (atrás)

Este proyecto es uno de los proyectos de investigación en el marco del ecosistema de proyectos de Robotito, que propone la incorporación de un celular con una aplicación móvil para expandir las formas de interacción con las que cuenta Robotito actualmente. La cámara del celular funciona como un nuevo sensor, y la pantalla y el sonido funcionan como actuadores. Adicionalmente la conexión a internet amplía las posibilidades de modificar el comportamiento del robot de forma remota. La capacidad de procesamiento de los celulares modernos también permite implementar lógicas más costosas en cuanto a cómputo, con un mejor desempeño, que las que se pueden ejecutar en la placa actual de Robotito. Finalmente agrega la posibilidad de desarrollar una interfaz gráfica. Debido a que al agregar el dispositivo móvil se generan todas estas nuevas formas de interacción, se considera que amplía las capacidades de Robotito a la hora de ser utilizado como robot educativo.

2. Revisión de antecedentes

Se revisó casos relevantes para el proyecto, empezando por un breve repaso de la historia de la robótica educativa y como fue su evolución en Uruguay, luego se estudiaron robots educativos que integran un celular en su sistema para utilizar su cámara, finalmente se elaboró un resumen del estado actual de robotito.

2.1. Robótica educativa

Los inicios de la robótica educativa se puede remontar a los primeros juguetes robóticos como Lilliput (Figura 2.a), un robot japonés desarrollado en la década del 30. En el año 1967 se desarrolla el lenguaje de programación LOGO [6], creado por el Instituto Tecnológico de Massachusetts (MIT), específicamente para el área educativa, con el que se introdujeron las primeras experiencias de robótica en el aula. Un ejemplo es la utilización de los robots Turtles[7] en el año 1969, que incluían comandos básicos de movimiento y la posibilidad de bajar o subir un lápiz para dibujar en el piso.

Más tarde aparecen los primeros kits robóticos, que por su capacidad de ensamblarlos y programarlos de distintas maneras, permiten un uso educativo muy interesante. Uno de los primeros fue el kit LEGO LOGO [8], que expandía las funcionalidades de LOGO para permitir manipular distintos motores, sensores y luces, que se podían encastrar de distintas formas. Algunos ejemplos de kits ampliamente utilizados en el área de la educación son el Lego Mindstorms EV3, mBot [9] y en Uruguay el kit Butiá (figura 2.d). Estos kits suelen incluir conjuntos de sensores (en general sensor de distancia, de luminosidad, de contacto y de colores), actuadores (motores para el desplazamiento, motores de brazos, leds), baterías, unidades de cómputo programables, carcasas y elementos estructurales.

Un caso particular de un kit que está especialmente diseñado para su utilización por niños de 1º a 3º año de escuela es el kit BRICKO (figura 2.b), desarrollado en la universidad Southern Denmark como proyecto académico. Cuenta con una placa en donde insertando distintas fichas se va modificando el comportamiento del robot. Existen fichas que hacen girar al robot, que lo mueven adelante, y fichas para crear loops, entre otras. De esta forma, niños muy chicos son capaces de programar el robot modificándolo físicamente para realizar distintas tareas.

Si bien los kits permiten una gran flexibilidad y potencial en el ámbito educativo, en general no resultan ser lo más adecuado para niños pequeños, debido a que es necesario contar con habilidades básicas de programación y muchas veces manipular piezas complejas, pequeñas o delicadas. Algunos kits logran solventar estas dificultades al ser modulares. Estos se componen de diversos bloques que se unen de forma fácil e intuitiva, y cada uno de estos bloques tiene sus comportamientos definidos. El comportamiento general del robot se elabora mediante la unión de diversos bloques. Esto evita que sea necesario programar o manipular piezas pequeñas para utilizarlos y resolver diversos problemas. Algunos

ejemplos de estos kits modulares son Cubelets [10] y Modi [11].

En Uruguay, la robótica educativa da sus primeros pasos en la década de los 90', donde se adquirió equipamiento para 72 escuelas y para proyectos de fin de año de la educación media técnica [12].

A partir del año 2007, se pone en funcionamiento el Plan Ceibal [13], que para finales de 2009 ya había entregado una laptop XO por niño en escuelas públicas. Esto supone un impulso en la inclusión de la tecnología en las aulas y de nuevos métodos de enseñanza. Durante este tiempo surge en la Facultad de Ingeniería UDELAR el proyecto Butiá [14], que proporciona un kit robótico con sensores y motores, especialmente diseñado para funcionar con las XO. El kit es de código abierto y funciona con herramientas de programación incluidas en las laptops: e-toys4, Scratch5, TortugArte y Turtlebot, las cuales utilizan lenguajes de programación de alto nivel que permiten que los niños puedan utilizarlos fácilmente. Desde entonces el kit Butiá (figura 2.d) comienza a distribuirse en liceos públicos de todo el país y se realizan talleres de capacitación para su utilización en las aulas.

En simultáneo y a partir de 2012, Ceibal comienza a realizar talleres y distribuir otros kits robóticos en todos los centros de enseñanza media, técnica superior y primaria de tiempo completo o extendido en el país [13]. Se entregaron kits Lego WeDo para enseñanza primaria, que incluyen un motor y dos sensores. Para la enseñanza media se entregaron Lego Mindstorm NXT (figura 2.c), que cuentan con dos motores y cuatro sensores, y son programados mediante Scratch, o TortuBots. En los centros de enseñanza técnica se distribuyeron kits Fishertechnic y Arduino. En los últimos años estudiantes de enseñanza media han participado de varios mundiales de robótica, logrando distinciones importantes. En 2019 Uruguay fue sede de la competencia mundial First Lego League, orientada para niños y jóvenes entre 4 y 16 años.



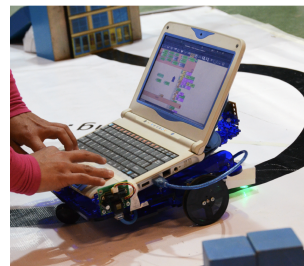
(a) Liliput [15]



(b) Brcko [16]



(c) Lego Mindstorm NXT [17], imagen tomada de [18]



(d) Butiá [14]

Figura 2: Robots educativos

2.2. Robots educativos con visión por celular

A partir de la popularización de los teléfonos inteligentes, el desarrollo de la robótica cuenta con una nueva herramienta para incluir en sus diseños. El utilizar celulares permite contar con capacidad de cómputo y memoria a bajo costo, además de distintos tipos de sensores y medios de comunicación: cámaras, micrófonos, sensores de contacto, acelerómetros, bluetooth, Wifi, etc.

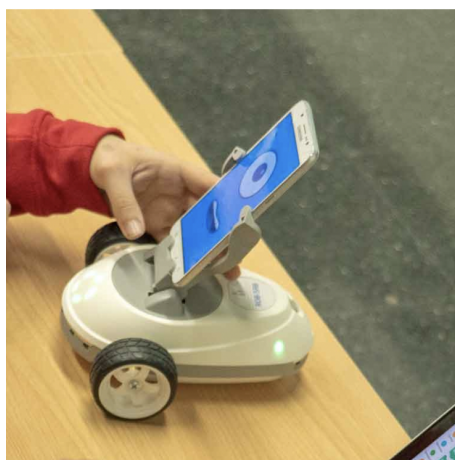
Un ejemplo de robot que se integra con un smartphone es el robot educativo Robobo (figura 3.a) , de la empresa MINT, se compone de una base con tres motores, que permiten mover dos ruedas y cambiar el ángulo de inclinación del celular. En el smartphone corre una aplicación que utiliza la cámara, permitiendo reconocer rostros, colores y evitar obstáculos. Usando el micrófono se puede utilizar reconocimiento de voz. Robobo permite programar sus comportamientos con diversos lenguajes de programación, (Scratch, Python, Java) y utilizar la plataforma ROS [19]. Se destaca la versatilidad que ofrece la posibilidad de programarlo en distintas plataformas, permitiendo así utilizarlo para diversos fines y diversos tipos de usuarios. Desde aquellos más experimentados en pro-

gramación, que podrán utilizar todas las herramientas y librerías que soportan Python, Java o ROS, a aquellos que tienen muy poca experiencia y son capaces de utilizar lenguajes gráficos de programación, o simplemente aquellos que pueden usar un robot con un comportamiento ya programado por un tercero, siendo posible de esta forma que sea utilizado por niños pequeños o usuarios de otras áreas no tecnológicas.

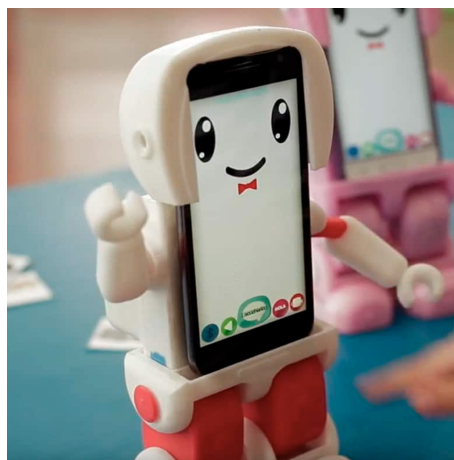
En Chile se han comenzado a utilizar los robots SIMA (figura 3.b) en las aulas. A diferencia de Robobo, estos robots cuentan con extremidades y una forma humanoide. Entre las funcionalidades del smartphone que utiliza se encuentra el reconocimiento de imágenes, permitiéndole detectar distintas tarjetas con dibujos, el reconocimiento de voz, y le permite utilizando el sistema Watson (IBM) elaborar conversaciones básicas. Del robot SIMA resulta interesante su capacidad para mantener conversaciones básicas, lo cual brinda un nuevo nivel de interacción entre los niños y el robot. Esto permite elaborar comportamientos basados en diálogos, por ejemplo acertijos, o conversaciones sobre los temas estudiados en el aula.

El robot SIRO [20] elaborado por AIBrain en Corea, se compone de una base con forma de auto, motores diferenciales y una aplicación para Android. Entre sus ventajas está la posibilidad de customizar su hardware agregando o quitando sensores o actuadores. Además de la aplicación, provee de un lenguaje para desarrollar aplicaciones propias llamado VPL. Es de interés para este proyecto, la posibilidad que ofrece de conectar otros sensores o actuadores robóticos y manipularlos desde el smartphone.

Uno de los ejemplos más exitosos es el proyecto Smart Robot Coding School [21], el cual se ha implementado como solución educativa en países como España, Francia, Brasil y Corea. Está orientado a la educación STEM (ciencia, tecnología, ingeniería y matemáticas) de niños de entre 4 y 12 años. Para implementarlo SK Telecom lanzó los robots educativos AlbertSchool (figura 3.d) y Albert UO, los cuales se componen de un cuerpo robótico, una aplicación para android y diversos elementos interactivos (dado y lapicera inteligentes, tablero de juego, cartas con dibujos). Ambas versiones de Albert pueden ser programadas utilizando el lenguaje Scratch. Es destacable para el proyecto la cantidad de experiencias en centros educativos que se han efectuado hasta el momento, lo cual es una muestra de la potencialidad en el área de la robótica educativa para robots que funcionen a base de smartphones.



(a) Robobo [22]



(b) Sima [23]



(c) Bandai Smartpet [24]



(d) Albert School [25]

Figura 3: Robots con visión por celular

En cuanto a juguetes también se han utilizado smartphones como componente para distintos tipos de robots. La empresa Bandai desarrolla smart pets robóticos (figura 3.c), que tienen apariencia de animales, donde en la cabeza se inserta un smartphone con una app. Es interesante que incluyen funcionalidades como detección e interacción con otros smart pets mediante bluetooth. Estas interacciones entre robots son útiles en contextos educativos, porque permiten ampliar las posibilidades de trabajar en áreas como el trabajo en equipo, la competitividad y el relacionamiento. Al funcionar como una mascota electrónica, está fuertemente orientado al entretenimiento.

En las tablas 1 y 2 se resumen algunas de las características de los robots analizados

	SIMA	Robobo	Bandai Smartpet
Forma / movilidad	humanoide / caminata	Ruedas / diferenciales	Animales / caminata
Actuadores	Movimiento de brazos	Inclinación y rotación del celular. Luces leds.	Motores para extremidades, cola y cabeza.
Sensores	No aparece en la documentación ni en videos de uso.	9 infrarrojo/luz ambiente, 4 de odometría (motores)	No aparece en la documentación ni en videos de uso.
Alimentación	Batería del smartphone.	Batería del smartphone.	Batería del smartphone y 3 pilas AAA.
Comunicación celular / base	Bluetooth	Bluetooth	Bluetooth
Sistema operativo	Android	Android	iOs
Reconocimiento de imagen	Tarjetas específicas	Facial (distancia entre ojos) Reconocimiento de colores	Objetos específicos (Hueso) Gestos y movimiento
Reconocimiento de voz	Preguntas y respuestas, reconocimiento contextual.	Comandos de voz	Comandos de voz
Programación de comportamientos	Básica (configuración de conversaciones)	Plataforma de desarrollo Scratch, Python, Java, Ros	No aparece en la documentación ni en videos de uso.
Sensores del smartphone utilizables	Cámara, micrófono, pantalla táctil, bluetooth.	2 Cámaras, luz temperatura, giroscopio, acelerómetro, magnetómetro. GPS, pantalla táctil, Wifi, Bluetooth, 3G, 4G.	Cámara, micrófono, pantalla táctil, bluetooth

Tabla 1: Comparación entre SIMA, Robobo y Bandai Smartpet. La documentación de SIMA y Bandai smartpet son incompletas o inexistentes, por lo que algunas características son asumidas en base a videos de la utilización práctica de los mismos.

	SIRO	Albert School	Albert UO
Forma / movilidad	Ruedas / oruga / diferenciales	Ruedas / diferenciales	Ruedas / diferenciales
Actuadores	LEDS	LEDS y 2 motores paso a paso (movilidad)	LEDS y 2 motores CC (movilidad)
Sensores	Personalizable, Distancia (Sonido, infrarrojo), Cámaras	Proximidad, Acelerómetro, Óptico de Identificación Luz, Temperatura, Batería baja.	Proximidad, Acelerómetro, Óptico de identificación Luz, Temperatura, Batería baja, tacto.
Alimentación	Batería del smartphone	Tipo Lipo Li-Polymer 3.7V (1600mAh)	Tipo Lipo Li-Polymer 3.7V (1600mAh)
Comunicación celular / base	No se encontró esta información.	Bluetooth 4.0 BLE	Bluetooth 4.0 BLE
Sistema operativo	Android	Android	Android
Reconocimiento de imagen	Cloud computing	Tarjetas, Tablero de juego, Reconocimiento de colores.	Tarjetas, Tablero de juegoReconocimiento de colores.
Reconocimiento de voz	Cloud computing	Ninguna	Ninguna
Programación de comportamientos	Programable con VPL	Scratch	Scratch

Tabla 2: Comparación entre SIRO, Albert School y Albert UO.

2.3. Robotito

A continuación se profundiza en el robot a ser utilizado en el proyecto, Robotito [4] es un robot diseñado con fines educativos, fue pensado para ser utilizado por niños y niñas, entre 4 y 7 años, con el fin de ayudar a desarrollar su pensamiento computacional. La idea que persigue Robotito es que los niños sean capaces de programar su comportamiento modificando su entorno. Dispone actualmente de dos formas de tomar características del entorno para determinar el comportamiento, la primera es mediante detección del color utilizando un

sensor que apunta al suelo y la segunda a través de sus sensores de distancia ubicados en sus costados.

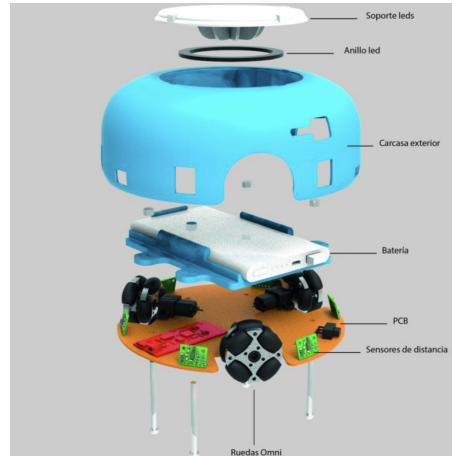


Figura 4: Componentes de Robotito

Los componentes del robot (figuras 4 y 5) fueron pensados de forma de abaratar su producción y aumentar su flexibilidad y adaptabilidad, eligiendo componentes ampliamente utilizados y estandarizados. Además, se tuvo un especial cuidado en la robustez para que sea adecuado para trabajar con niños.

Sus componentes están protegidos por una carcasa de plástico de forma circular, la cual se compone de una pieza impresa en 3d. Cuenta con un microcontrolador Sparkfun ESP32 Thing el cual brinda el CPU, RAM y almacenamiento necesario, así como también puertos de entrada y salida para comunicarse con los sensores y actuadores, e interfaces de Wifi y Bluetooth. Su movilidad la realiza mediante tres ruedas omnidireccionales, orientadas a 120° entre sí, lo cual permite movilidad holonómica [5]. Las ruedas son potenciadas por motores CC estándar con reducciones integradas. Los sensores que Robotito tiene disponibles son, una serie de seis sensores láser VL53L0X, que utilizan time-of-flight para medir la distancia a posibles objetos, y están ubicados alrededor de Robotito cada 60° . El otro sensor del que dispone es un APDS-9960, ubicado apuntando al piso en el centro del robot, que se encarga de sensar los posibles colores del suelo y su proximidad. Finalmente, en la parte superior tiene un anillo LED NeoPixel, que se usa para comunicar información al usuario.

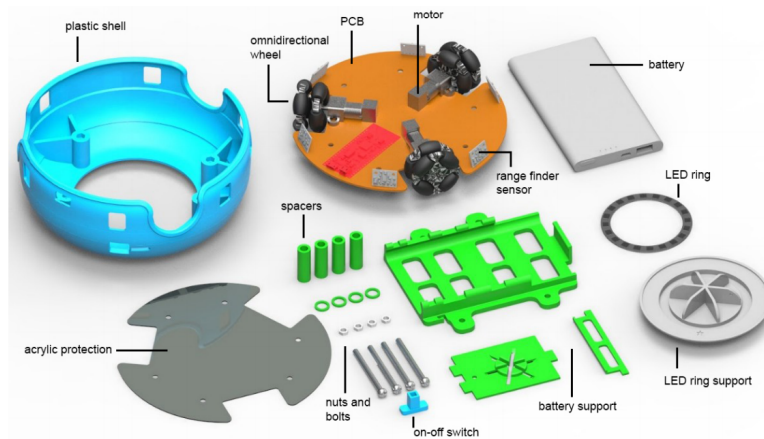


Figura 5: Componentes de Robotito

El firmware de Robotito fue programado utilizando el lenguaje Lua, y utiliza el paradigma de programación orientada a eventos. Su programa principal está implementado como una máquina de estados jerárquica [5], que define un conjunto de reglas de cómo el robot se debe comportar ante las entradas de sus sensores.

Robotito cuenta con tres formas de comportarse por defecto, siendo posible crear otras. La primera de ellas, basada en su sensor de color, permite que reaccione cambiando de dirección según los colores detectados, utiliza cinco colores: rojo, azul, verde, amarillo y violeta. Cada color tiene asignado una dirección y un sentido en que se moverá Robotito al detectarlo. De esta forma, los niños colocan parches de colores en el piso que definen cómo se moverá Robotito. El segundo modo, utiliza los sensores de distancia para detectar los objetos que se encuentran alrededor, y se mueve como si los objetos le aplicaran una fuerza de atracción proporcional a la distancia a la que se encuentran. En este modo, para controlar el robot se colocan objetos de forma de ir desviando su trayectoria en la dirección deseada. Finalmente, un tercer modo recibe comandos por Wifi y ejecuta el movimiento deseado.

Robotito es una herramienta educativa, que ha permitido experiencias interesantes y didácticas, y cuenta con proyectos de investigación que lo utilizan de distintas formas [26]. Es por eso que este proyecto aporta a las actividades de investigación que se crean alrededor de Robotito, en este caso integrando una aplicación móvil. Se busca así, agregar nuevas formas de interacción con Robotito, que permitan elaborar dinámicas educativas más variadas, que incentiven a alumnos y educadores. Un dispositivo móvil aporta una variedad de elementos en ese sentido. La cámara permite aumentar la cantidad de elementos que se pueden detectar del entorno, la pantalla puede mostrar imágenes y textos que enriquezcan las dinámicas, y los parlantes permiten explorar otro sentido en cuanto a la interacción.

Además de los aportes en cuanto a la interacción, la inclusión de un celular provee nuevas herramientas para la distribución y actualización, ya que la aplicación podría ser fácilmente descargada, instalada y actualizada cuando sea necesario, facilitando la tarea de preparar a Robotito para las actividades. Finalmente, la capacidad de cómputo que aporta un celular permite manejar más cantidad de datos en algoritmos más complejos, lo cual amplía las capacidades de Robotito.

3. Diseño de solución

En esta sección se detalla el diseño de la solución implementada, explicando los factores que se tomaron en cuenta, los usuarios potenciales identificados, el paradigma utilizado, y los objetivos buscados con el diseño. Luego se especifican las funcionalidades propuestas y consideraciones generales para la arquitectura del sistema. Finalmente, se muestra la arquitectura tomando en cuenta los puntos anteriores.



Figura 6: Robotito con el dispositivo móvil incorporado, utilizando como soporte un prototipo de cartón.

Tomando como punto de partida el objetivo del proyecto, se diseñó una solución que se basa en una aplicación a ser ejecutada por un dispositivo móvil ubicado en la parte superior de Robotito (figura 6).

Para el diseño de la solución se tuvo en cuenta a los tipos de usuarios identificados:

- Los desarrolladores e investigadores: usuarios técnicos, que podrían buscar incorporar cambios en el software o en parámetros del mismo, por lo que la aplicación deberá ser flexible y fácil de adaptar.
- El personal de la educación: usuarios no técnicos, que deberían ser capaces de utilizar la aplicación y de esta forma controlar las dinámicas educativas, de modo que la aplicación deberá ser fácil de usar, con una interfaz simple.
- Los niños, quienes interactuarán con el entorno y Robotito, para quienes es necesario pensar nuevas formas de interacción.

En la robótica autónoma existen diversos paradigmas para modelar los comportamientos de los robots. Generalmente se agrupan en: reactivos y jerárquicos [27]. Se diferencian en que los robots con modelos jerárquicos planifican sus acciones realizando un modelo del entorno, mientras que los reactivos se saltan

esa etapa y según lo que detectan con sus sensores actúan de una u otra manera. En el caso de Robotito, donde se busca que los niños puedan asociar rápidamente cambios en el entorno a cambios en cómo actúa Robotito, se utiliza un paradigma reactivo. En particular se modelan los comportamientos como máquinas de estados jerárquicas, implementadas en su firmware. Para este proyecto se decidió mantener este paradigma y utilizar máquinas de estados. Esto permitirá a los proyectos que ya están en funcionamiento, la posibilidad de comenzar a usar un dispositivo móvil sin efectuar cambios de paradigma.

Se entiende una máquina de estados finita como: “Un grafo dirigido, cuyos nodos representan estados del sistema y sus aristas corresponden a transiciones del sistema” [28] Para la solución propuesta se propone utilizar simultáneamente varias máquinas de estados estableciendo un sistema de prioridad sobre el uso de los recursos que se utilizan en cada estado, por lo que se utilizará el término “Máquinas de estado concurrentes”.

Un aspecto que se consideró en el diseño de la solución, es mejorar el despliegue de nuevas versiones y funcionalidades de Robotito. Actualmente, los investigadores al trabajar con Robotito, si quieren cambiar un parámetro o modificar la máquina de estados que define su comportamiento, deben tener una comprensión del código en Lua, para saber cómo programar la nueva máquina de estados o dónde cambiar el parámetro, y finalmente copiar los archivos en cada Robotito en el que se desee disponer la nueva versión. Al incluir una aplicación móvil y un celular, se propone subir un nivel de abstracción y que esos cambios puedan ser aplicados sin manipular físicamente a Robotito, mediante actualizaciones de la aplicación o sus datos remotamente.

Se buscó que la solución diseñada permitiera a Robotito ampliar sus posibilidades como plataforma, es decir, que se promoviera el desarrollo de nuevos comportamientos del robot por los usuarios. Esto permite que los usuarios encuentren e implementen casos de uso que los desarrolladores no imaginaron. La flexibilidad y las posibilidades que ofrece una plataforma es algo que se buscó contemplar para colaborar en potenciar el ecosistema del robot. En cuanto a la capacidad de combinar estas herramientas para elaborar comportamientos, sería deseable que se pudiera realizar a alto nivel, sin necesidad de programar, ya que una solución así acercaría la posibilidad de que usuarios no técnicos pudieran comenzar a elaborar comportamientos de Robotito.

Teniendo en cuenta estas consideraciones, se propone crear una aplicación móvil que pueda funcionar como una plataforma y presente un conjunto de herramientas con foco en el reconocimiento de imágenes. Los investigadores o desarrolladores podrán crear comportamientos para Robotito (figura 7) y almacenarlos en una base de datos en la nube. Cada comportamiento estará implementado como máquinas de estados concurrentes, que definirán cuándo y cómo actúa el robot. Luego la aplicación descargará o sincronizará esos comportamientos para ser utilizados. La aplicación será la encargada de percibir el entorno del robot, para decidir en base al comportamiento en ejecución qué acción debe tomar Robotito en cada momento. Finalmente los niños o alumnos manipularán el entorno, cambiando lo que percibe la aplicación y afectando de esta forma el accionar de Robotito.

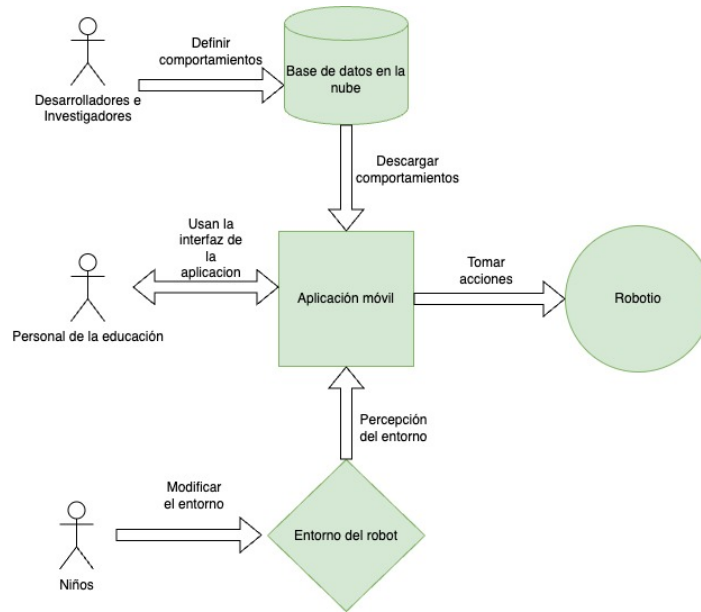


Figura 7: Diagrama de uso de la aplicación

A continuación se detallan funcionalidades, características generales y arquitectura general de la aplicación.

3.1. Funcionalidades propuestas

Teniendo en cuenta la propuesta de solución, se estudió mediante prototipado la factibilidad de posibles funcionalidades. Los prototipos realizados permitieron mitigar riesgos relacionados con el reconocimiento de imágenes en un dispositivo móvil con bajas prestaciones. Además, sirvieron en el análisis de tecnologías y permitieron detectar y descartar una funcionalidad que presenta complejidades para el alcance de este proyecto (reconocimiento de poses). Finalmente, se propuso las siguientes funcionalidades:

3.1.1. Definir un comportamiento

Se propone que usuarios técnicos (desarrolladores o investigadores) puedan definir comportamientos de Robotito, los cuales serán definidos como máquinas de estados concurrentes.

Los comportamientos serán definidos a través de un manejador de base de datos y serán alojados en una colección en la nube, la cual se deberá sincronizar automáticamente con la colección que contiene cada instancia instalada de la aplicación móvil. Para definir el comportamiento de Robotito, es necesario establecer sus estados, los recursos que se utilizan en cada uno y bajo qué condiciones cambia de estado. En definitiva, determinará las diferentes formas en las

que actuará, dependiendo de la interacción con el entorno o el paso del tiempo. Por ejemplo, Robotito podría estar en un estado de “buscar un cubo rojo”, en el cual usa como recurso sus motores para girar en el lugar hasta que se cumpla la condición de detección del cubo rojo, momento en el que se detendrá.

De cada estado se registra una descripción, sus recursos y sus transiciones (las condiciones que lo llevarán a otro estado).

Los recursos que podrán utilizarse son:

- El movimiento del robot.
- La pantalla del dispositivo móvil.
- Salidas de sonido del dispositivo móvil.

Las transiciones que podrán ser utilizadas para cambiar de estado son:

- Al transcurrir cierto tiempo.
- Al detectar marcas, determinados objetos o figuras geométricas específicas en su entorno.

Además, cada transición definirá en qué estado quedará Robotito al cumplirse su condición. Siguiendo con el ejemplo anterior, se podría definir un estado en el que Robotito rote hacia la derecha buscando un cubo rojo (figura 8), una condición de que cuando detecte el cubo rojo despliegue en pantalla un cubo rojo, emita un sonido por un segundo y pase a un siguiente estado en el que buscará un cubo azul rotando hacia la izquierda, y que al detectarlo vuelva al estado inicial.

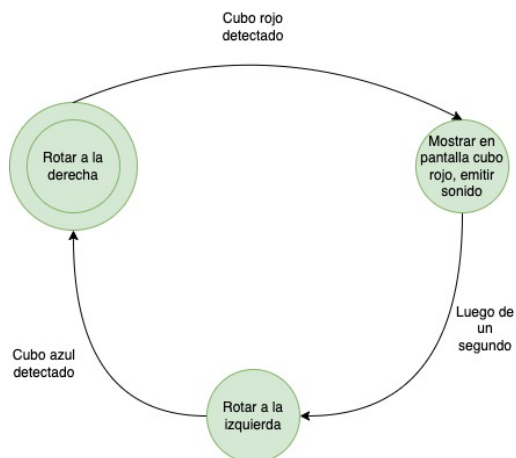


Figura 8: Máquina de estados de un comportamiento de ejemplo, donde el robot debe rotar a la derecha hasta detectar un cubo rojo y luego a la izquierda hasta detectar un cubo azul.

3.1.2. Seleccionar un comportamiento

Se propone que los usuarios de la aplicación móvil dispongan de una colección de comportamientos y que sean capaces de elegir uno desde la interfaz para seleccionarlo como el comportamiento actual de Robotito. De esta forma, al permitir cambiar de comportamiento, se puede disponer de varios y que cada uno se ajuste a la dinámica que se planea realizar en el aula.

La aplicación mostrará cuál es el comportamiento actual de Robotito. Una vez que el usuario presione el nombre del comportamiento actual, se desplegará un listado de comportamientos posibles, el usuario seleccionará uno y el comportamiento actual de Robotito cambiará para el seleccionado.

3.1.3. Control remoto

Los usuarios tendrán la posibilidad de manejar a Robotito por control remoto mediante un joystick en pantalla.

La aplicación desplegará el nombre del comportamiento activo, si el usuario le hace click se listarán los comportamientos disponibles en la colección y se mostrará un comportamiento específico llamado “control remoto”. Al seleccionarlo se mostrará en pantalla un joystick que controlará el movimiento traslacional de Robotito, dos botones que controlarán la rotación del mismo (horario y antihorario), un tercer botón que detendrá todo movimiento.

3.1.4. Detección de marcas

Se considera que la detección de marcas es una funcionalidad interesante para sumar a las capacidades de Robotito, ya que es una tecnología madura y confiable, que permite que se codifique texto en una marca. Android cuenta con soluciones nativas para detectarlo, que funcionan correctamente.

Se propone que la aplicación móvil sea capaz de detectar mediante la utilización de la cámara trasera del dispositivo, las diferentes marcas que se encuentran en su campo de visión. Las marcas serán decodificadas y se evaluará si corresponde que Robotito cambie de estado. Además, se desea que sea capaz de localizar en qué parte de la imagen se encuentran. La definición del comportamiento en curso será la que guíe la forma en la que Robotito reaccione a las diferentes marcas que pueda detectar en su campo de visión.

Por ejemplo, si Robotito estuviera en un estado: “Girando hasta encontrar un código QR que diga ‘Pelota’”, se desea que reaccione ante códigos QR que tengan codificado el texto “Pelota”, cambiando de estado y deteniéndose.

3.1.5. Etiquetado de imágenes (general)

Se propone que la aplicación móvil tenga la capacidad de utilizar la cámara trasera del dispositivo, para detectar si ciertos objetos predefinidos se encuentran o no en su campo de visión. Se desea que la lista de objetos detectables sea lo más general posible de forma de disponer de cierta flexibilidad en su uso.

Al detectar que en la imagen se encuentra cierto objeto buscado, dependiendo de la definición de el comportamiento en curso, se decide de qué forma debe reaccionar Robotito. Por ejemplo, la definición del comportamiento podría especificar que “al detectar una mano, se debe detener todo movimiento”.

3.1.6. Etiquetado de imágenes (figuras geométricas)

Para el etiquetado de imágenes para objetos generales, existen limitaciones tecnológicas de trabajar en tiempo real con objetos para los que la aplicación no fue específicamente preparada. Debido a esto se plantea agregar un etiquetado de imágenes para objetos con los que esté específicamente entrenado para detectar.

La aplicación móvil será capaz de, mediante la cámara del dispositivo, detectar si ciertas figuras geométricas particulares se encuentran en su visión. Cuando una de estas figuras geométricas es detectada de esta manera, dependiendo lo que se haya definido en el comportamiento en curso de Robotito, se decide la forma en la que Robotito reaccionara.

Por ejemplo, si Robotito se encuentra en un estado: “Girar a la izquierda al detectar un cubo azul”, al detectar la aplicación un cubo azul, Robotito deberá girar a la izquierda.

Las figuras que se buscan detectar son: cubos, pirámides de base cuadrada y esferas, de colores rojo, azul o verde.

3.1.7. Reproducir sonidos

La aplicación móvil tendrá la posibilidad de reproducir sonidos, utilizando las salidas de audio del dispositivo (parlantes, auriculares, etc). La aplicación dispondrá de una colección de sonidos disponibles que podrán ser elegidos al definir los recursos que utilizará un comportamiento, de forma que cuando Robotito tenga un comportamiento en curso, y éste utilice un sonido, el mismo sea reproducido por el dispositivo.

Por ejemplo, si en el comportamiento en curso se está en un estado: “Reproducir sonido de trompetas cuando se detecte el código QR con el texto codificado: ‘trompetas’”, al aparecer dicha marca el dispositivo emitirá el sonido de trompetas.

Se desea que la librería de sonidos disponga de sonidos musicales, y que sea accesible agregar o cambiar sonidos en las siguientes versiones de la aplicación.

3.1.8. Desplegar imágenes/gifs en pantalla

La aplicación móvil tendrá la posibilidad de reproducir imágenes o gifs en pantalla. Se dispondrá de una colección de imágenes y gifs que podrán ser elegidos al momento de definir los recursos de un comportamiento. Cuando se tenga un comportamiento en curso, al utilizar como recurso la pantalla del dispositivo, se desplegará la imagen especificada en el comportamiento.

Se deberán contar con gifs disponibles para cada forma geométrica detectada, y se desea, que sea fácil agregar nuevas imágenes o gifs en siguientes versiones de la aplicación.

3.2. Consideraciones generales

La adaptabilidad y flexibilidad de la aplicación son características claves, y se las consideraron en los siguientes aspectos:

- Modificar los comportamientos desde el manejador de base de datos y que los cambios impacten en las instancias de la aplicación, sin la necesidad de que un usuario especializado manipule cada dispositivo móvil, ni qué sean necesarios cambios de versión en la aplicación.
- Es posible agregar nuevos recursos, sea agregando gifs, sonidos, o creando tipos de recursos nuevos que no estén contemplados (por ejemplo, en un futuro se podría agregar la posibilidad de incluir el uso de las luces leds en las máquinas de estados)
- Elementos detectables por la cámara. Se busca que sea posible agregar nuevos elementos modificando el código, y liberando una nueva versión.

Por otro lado se busca que la interfaz gráfica no sea compleja, y que la aplicación sea utilizable por usuarios no técnicos, sean maestros, profesores o alumnos. Otra característica importante es que la aplicación deberá poder trabajar en tiempo real, por lo que los tiempos de procesamiento se deberán mantener acotados. Si al manipular el entorno Robotito reacciona tarde, dificulta el entendimiento de los usuarios de cómo Robotito reacciona a su entorno.

Una limitante que cuenta el presente proyecto, es que actualmente se utiliza el Wifi del celular para establecer la conexión con Robotito, lo cual implica descartar posibles funcionalidades como el procesamiento en la nube de modelos de reconocimiento de imágenes. Una opción para palear esa limitante podría ser cambiar el firmware de Robotito de forma de establecer la conexión con el celular a través de bluetooth, de forma de liberar su conexión Wifi para conectarse a internet. Si bien puede ser una posibilidad en un futuro, se considera que para el alcance de este proyecto se prefiere una aplicación que pueda funcionar sin conexión a internet, brindando de esta forma una mayor flexibilidad a su uso en entornos donde no se disponga de conexión. Además se considera que hacer el procesamiento en la nube de las imágenes de la cámara implica un mayor riesgo de seguridad, y dado que en estas imágenes aparecerían niños se considera mejor que el procesamiento se realice en el dispositivo.

3.3. Arquitectura general

Se propone dividir la arquitectura en subsistemas, cada uno de los cuales contará con una función, de forma de facilitar la legibilidad, el mantenimiento y la reutilización del código.

Los subsistemas identificados son:

Interfaz gráfica: Permite seleccionar entre distintos comportamientos de Robotito. Para el comportamiento especial de “Control Remoto” despliega un joystick. Permite pausar o reproducir los comportamientos y en caso de ser necesario desplegar imágenes/gifs en pantalla o reproducir sonidos.

Lógica de comportamientos: Es el módulo central de la aplicación. Obtiene la definición del comportamiento seleccionado desde la base de datos, y lo ejecuta, controlando en qué estado del comportamiento está en cada momento, según los datos que recibe del módulo procesador de imágenes. A su vez, se encarga de enviar comandos a Robotito para utilizar sus motores según el estado actual de su comportamiento.

Cámara: Es el módulo encargado de capturar las imágenes a través de la cámara del dispositivo.

Procesador de imágenes: A partir de las imágenes capturadas por la cámara, se encargará de reconocer los elementos deseados del entorno de Robotito. Esto es detección de marcas, etiquetado de imágenes (figuras geométricas) y etiquetado de imágenes (general) y envía lo procesado al módulo: Lógica de comportamientos.

Bases de datos: Son las encargadas de almacenar las definiciones de comportamientos. Debido a la limitante de que el dispositivo utilice el Wifi para comunicarse con Robotito, es necesario pensar en un módulo de datos que pueda funcionar sin conexión a internet. Por lo que se requiere una base de datos local (en el dispositivo), y una en la nube (necesaria para la funcionalidad de definir un comportamiento), las cuales se sincronizan cuando la aplicación tiene conexión a internet.

Robotito: La aplicación se comunica con el firmware de Robotito, quien será el encargado de manipular los motores del robot.

El siguiente esquema (figura 9) representa cómo se comunican los subsistemas identificados. Además el diagrama representa cuáles de los subsistemas se implementarán como parte de la aplicación y cuáles funcionarán como sistemas externos

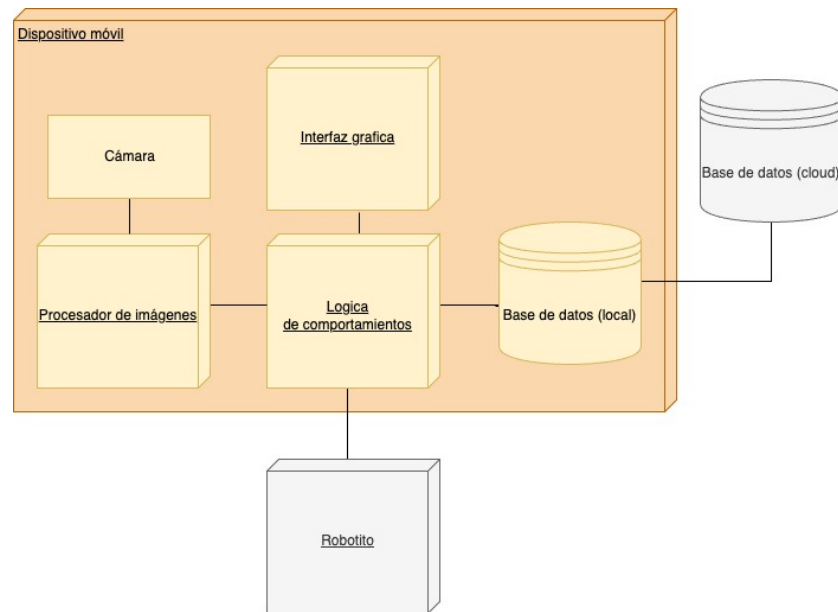


Figura 9: Arquitectura general de la aplicación, subsistemas internos y externos

4. Implementación

A continuación se detallara el análisis de tecnologías realizado, así como las decisiones que se tomaron al respecto. Luego se detallara la arquitectura implementada, se profundizara en el funcionamiento de los comportamientos y los modelos utilizados para reconocimiento de imágenes.

Análisis de tecnologías

Para elaborar el diseño y la arquitectura general de la aplicación, se optó por comenzar definiendo las tecnologías a utilizar, ya que estas impondrán restricciones sobre el diseño que es necesario detectar en etapas tempranas del proyecto.

4.0.1. Plataforma

Una primera decisión en cuanto a las tecnologías fue optar por desarrollar una aplicación para la plataforma Android (Google, 2013), ya que es la más utilizada en Uruguay [29], y se busca que sea accesible contar con un celular que pueda ejecutar la aplicación. En contraposición con IOs (Apple, 2007) que es la otra plataforma que cuenta con gran popularidad, Android ofrece una mayor flexibilidad, tanto para el desarrollo (que en el caso de IOs es necesario contar con una computadora con el sistema operativo Mac OS X), como para la distribución de la aplicación, donde la App Store (el servicio donde se publican las aplicaciones para IOs) tiene políticas de aceptación más estrictas que Google Play, su contraparte de Android. Otra ventaja que genera la popularidad de Android (figura 10), es que al ser más utilizada cuenta con más documentación, paquetes, librerías, y ayuda en línea.

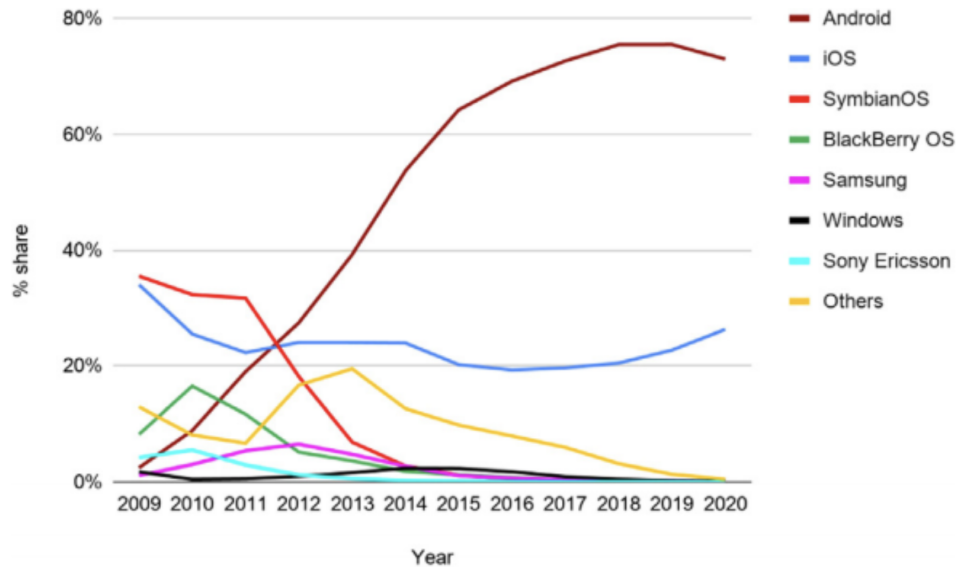


Figura 10: Participación de mercado de los sistemas operativos móviles a través de los años. [30]

4.0.2. Framework de desarrollo

El framework de desarrollo que se eligió es React Native (Meta, lanzado en 2015), el cual permite escribir en el lenguaje Javascript y renderizar en Android Nativo. Otras posibilidades evaluadas fueron, utilizar directamente Android Nativo sin un framework (utilizando el Android SDK, en lenguaje Java), o usar otro framework como Flutter (lanzado en 2017 por Google, se programa en el lenguaje Dart), Ionic (creado por Drifty Co. en 2013, permite programar en Javascript, HTML5 y CSS3), o Apache Cordova (Nitobi, 2009). Se destaca que todos los frameworks analizados son de código abierto.

Para decidir qué framework utilizar entre los mencionados, se analizaron dos factores principales. En primer lugar, los lenguajes de programación en los que permite desarrollar, donde se priorizó frameworks de Javascript debido a que se contaba con mayor experiencia en ese lenguaje. En segundo lugar, la popularidad del framework, que es importante debido a que en los frameworks más populares es más posible encontrar documentación, foros de discusión, librerías y soporte, además permiten una mejor reutilización del proyecto si este está implementado en una tecnología muy utilizada.

De los frameworks analizados, aquellos que permiten programar en Javascript son: Apache Cordova (versión de PhoneGap en código abierto), React Native y Ionic (construido en base a Apache Cordova). La ventaja más importante que tiene React Native sobre los otros dos frameworks, son las facilidades que brinda para utilizar elementos nativos del dispositivo, cosa que Ionic y Apache

Cordova solo permiten a través de plugins, los cuales pueden provocar latencias, lo cual es un factor decisivo en una aplicación en tiempo real que utiliza principalmente la cámara [31]. Para analizar la popularidad de los frameworks se tomó en cuenta una encuesta (figura 11) que se realizó globalmente entre 2019 y 2021, donde se preguntaba a desarrolladores qué framework utilizan.

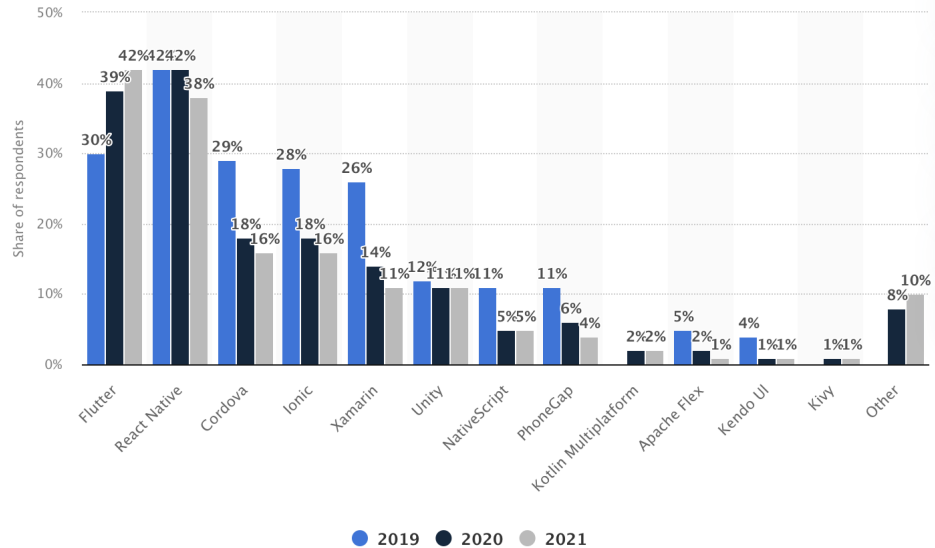


Figura 11: Encuesta de popularidad de frameworks de desarrollo de aplicaciones móviles[32].

Si bien la tendencia analizada mostraba a Flutter como un gran competidor, cuando se inició el presente proyecto, React-Native era el framework con mayor popularidad. Debido a que no se contaba con experiencia en el lenguaje Dart y a que React Native facilita la utilización de elementos nativos de los dispositivos, se decidió utilizar React Native.

A pesar de que se decidió implementar la mayoría del código en React Native, algunas funcionalidades requieren que se programe directamente en Android Nativo, debido a que exigen bajos tiempos de ejecución para algoritmos complejos de reconocimiento de imagen, y un control sobre el preview de la cámara que no ofrecían ninguna de las bibliotecas para manejar la cámara en React Native. En esos casos React Native permite crear componentes en Android y utilizarlos directamente en React, por lo que no representó un riesgo importante.

4.0.3. Modelos de reconocimiento de imágenes

Se estudiaron dos alternativas de modelos de reconocimiento de imágenes especializados para dispositivos móviles. Core ML, desarrollado por Apple en

2017, es un framework de machine learning que permite crear modelos para ser utilizados en dispositivos móviles. Se descartó debido a que solo funciona para iOS. La otra alternativa, TensorFlow Lite, elaborado por Google y lanzado en enero de 2019, es una plataforma de código abierto que facilita la creación y manipulación de modelos de aprendizaje automático. Permite, entre otras cosas, elaborar y entrenar redes neuronales para el reconocimiento de patrones. Su versión Lite está especialmente diseñada para trabajar en dispositivos con capacidad de cómputo limitada. Al fijar que se utilizarían los modelos de TensorFlow Lite, resultó inmediata la integración de varias herramientas que ayudan a su manipulación, almacenamiento y uso. Estas herramientas se comunican entre ellas y al ser desarrolladas por la misma empresa, son soluciones compatibles y ya probadas.

4.0.4. Firebase

Firebase (Google, 2011) es una plataforma de desarrollo de aplicaciones móviles que brinda herramientas de hosting, autenticación, bases de datos, storage, machine learning y analytics entre otras. Algunas de las cuales son de gran utilidad para resolver las funcionalidades propuestas. Entre sus herramientas de machine learning, permite alojar modelos de Tensorflow Lite en la nube, para descargarlos y utilizarlos en el dispositivo. En caso de modificar un modelo personalizado alojado, el modelo es actualizado en todas las instancias de la aplicación apenas dispongan de internet. Tanto los modelos de la API, como los propios, pueden ser utilizados desde el dispositivo.

Otra de las funcionalidades destacables de Firebase es que ofrece varias alternativas como base de datos. En particular, Realtime Database permite almacenar datos en una base de datos NoSQL, y mantenerla sincronizada con bases de datos en todos los clientes de la aplicación, además de que cada cliente puede acceder a su base de datos sin tener conexión. La ventaja de Realtime Database es la flexibilidad que brinda. Permite que si se desean guardar nuevos datos no sean necesarios cambios de esquemas, y que los cambios repercutan en todas las instancias de la aplicación. Esta funcionalidad resolvería tanto la necesidad de una base de datos local como la de una en la nube, y su capacidad de sincronización permite solucionar las discrepancias en los datos que se podrían dar por mantener dos bases de datos.

4.0.5. MLKit

El kit de machine learning MLkit, desarrollado por Google y lanzado en mayo de 2018, provee herramientas que resultan útiles para el proyecto. Dispone de una API que permite resolver algunos de los usos comunes de machine learning, entre los que se encuentra el reconocimiento de marcas y etiquetado de imágenes (para objetos genéricos). También ofrece la posibilidad de utilizar modelos de Tensorflow Lite propios, lo cual aporta mucha flexibilidad a lo que se puede hacer con el kit. Permite además la posibilidad de integrarse con Firebase fácilmente. Tanto para los modelos que ofrece la API como para los personalizados, permite

ejecutarlos en el dispositivo o en la nube, siendo totalmente gratuito el uso en el dispositivo, mientras que para la ejecución en la nube de los modelos se cobra a partir de cierto uso. Se decidió utilizar MLkit ya que en este proyecto se necesita poder ejecutar los modelos en el dispositivo, y MLkit permite cargar modelos propios localmente. Esta necesidad de ejecutar los modelos en el dispositivo viene dada porque no siempre se contará con internet para ejecutar modelos en la nube y que, además, ejecutando en el dispositivo se tiene un mejor control de qué modelo se está ejecutando y qué se hace con los datos.

4.0.6. Google Cloud

Google Cloud (desarrollado por Google, en 2008) es una plataforma que ofrece un abanico de soluciones y servicios tecnológicos para el desarrollo, administración, cloud computing, almacenamiento, bases de datos, análisis de datos, IA y aprendizaje automático, redes y otras herramientas de desarrollo. Entre las muchas funcionalidades en cada una de esas áreas, la que nos interesa es la de elaborar nuestros propios modelos de TensorFlow Lite, a partir de un dataset a elección. Permite crear modelos para distintos problemas (lenguaje natural, etiquetado de datos, etc), pero el más relevante para el proyecto es su producto de “Visión”, que se especializa en etiquetado y clasificación de imágenes. Google Cloud se puede utilizar integrada con Firebase y Tensorflow, siendo las tres desarrolladas por Google. Se considera eso una ventaja, ya que trabajar con un stack de tecnologías de la misma empresa implica en general que no va a haber incompatibilidades entre ellas.

El siguiente diagrama (figura 12) ilustra cómo se propone organizar el stack de tecnologías, cuáles se utilizarán y cómo se comunican entre ellas:

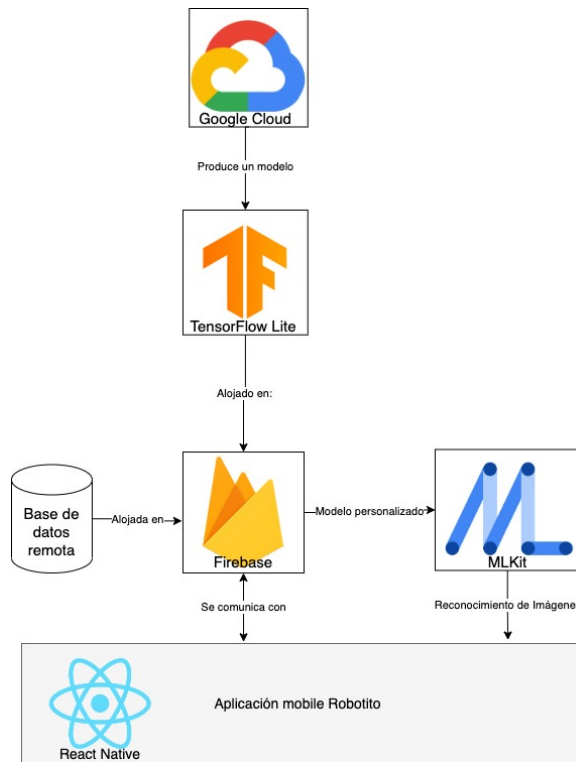


Figura 12: Stack de tecnologías utilizadas

La aplicación móvil de Robotito será implementada en React Native, desarrollada para Android. Se utilizarán las capacidades de Google Cloud para entrenar modelos de Tensorflow Lite de reconocimiento de imagen que sean útiles para Robotito, en particular un modelo de reconocimiento de figuras geométricas. El modelo generado se alojará en Firebase, desde donde se manejan sus versiones. Además, se utilizará la base de datos “real time database” que ofrece Firebase, por lo que la aplicación deberá comunicarse con Firebase para sincronizar su base de datos local con la de Firebase. Finalmente se utilizarán de MLKit sus API de visión: Barcode scanning e Image labeling, además de su funcionalidad para cargar modelos prediseñados. En este último caso se descargará el modelo de reconocimiento de figuras geométricas hospedado en Firebase.

4.1. Arquitectura implementada

Se decidió implementar una arquitectura en tres capas:

Capa de presentación: Está compuesta por las interfaces gráficas de la aplicación.

Capa lógica: Encargada de implementar las operaciones del sistema y sus re-

glas. Entre sus responsabilidades está decidir el comportamiento de Robotito.

Capa de datos: Encargada de comunicarse con las bases de datos.

Este modelo arquitectónico es una forma de organizar el código de forma clara y simple. Además, permite que en futuros desarrollos sobre el proyecto se pueda reemplazar una capa sin modificar el resto, lo cual, en un proyecto con fines de investigación es una característica deseable, ya que permite a los investigadores la flexibilidad necesaria para modificar la aplicación fácilmente.

Cada capa se separa en componentes de forma de dividir el código en módulos, cada uno con un propósito específico. Mediante el encapsulamiento de los componentes se apunta a mejorar la legibilidad del código y prever futuros cambios, esto debido a los requerimientos de flexibilidad mencionados anteriormente.

4.1.1. Componentes

En la figura 13 se presenta un diagrama ilustrativo de la arquitectura del sistema:

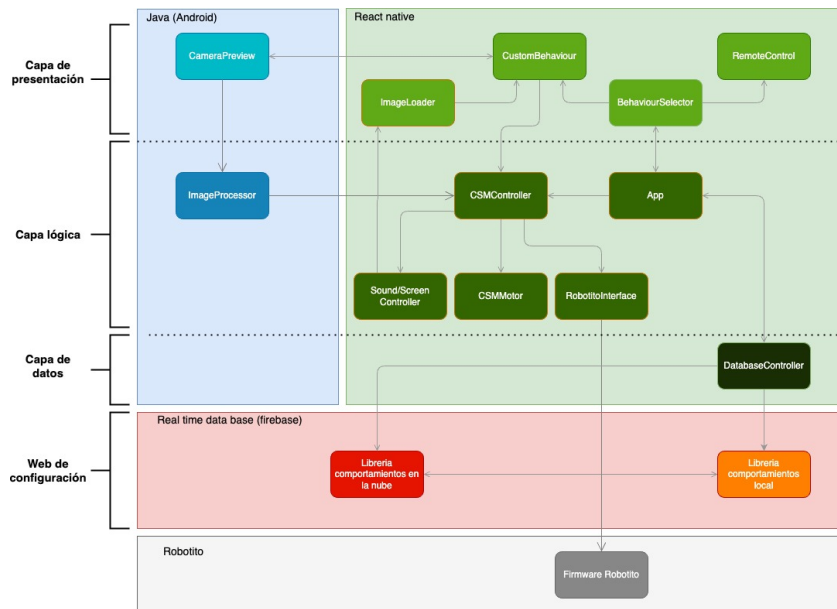


Figura 13: Diagrama de arquitectura, muestra los diferentes componentes, la separación en capas y las tecnologías utilizadas

En la figura 13 se pueden observar las diferentes capas de la aplicación (a la izquierda) mencionadas anteriormente, sumadas a una capa física de Robotito y a una web de configuración utilizada para manejar la base de datos remota.

Además, se dividieron los componentes según la tecnología en la que se implementaron. En verde se muestran los componentes desarrollados en React Native y en azul aquellos implementados en Java nativo. La web de configuración y Robotito son sistemas externos a la aplicación que se muestran en rojo y gris, respectivamente. Finalmente, la comunicación entre componentes se representa mediante flechas que los conectan.

Capa de presentación Dentro de la capa de presentación se encuentran todos los componentes encargados de mostrar una interfaz gráfica al usuario y de capturar sus interacciones. A excepción del componente de la cámara, todos fueron implementados utilizando React Native.

CustomBehaviour: Es el componente utilizado cuando se selecciona un comportamiento diseñado por un usuario. Muestra en toda la pantalla el componente de la cámara (figura 14.a), y sobre ese componente coloca un botón para detener o comenzar la ejecución de un comportamiento.

CameraPreview: Este componente implementado en Android Native, muestra en tiempo real el preview de la cámara trasera del dispositivo (figura 14.a) .

RemoteControl: Despliega un joystick en pantalla (figura 14.b), el cual se compone de un botón rojo que representa una palanca analógica, el cual permite al moverlo controlar la velocidad de Robotito en el eje “x” e “y” (adelante, atrás, izquierda, derecha). Además, contiene un botón rojo para detener todo movimiento, y dos botones amarillos para controlar el giro en torno al eje “z” (sobre sí mismo) en sentido horario o antihorario.

BehaviourSelector: Se encarga de permitir al usuario cambiar entre pantallas y muestra una barra superior en donde el usuario puede seleccionar un comportamiento. Al presionar el nombre del comportamiento actual, se despliega un diálogo con la lista de comportamientos (figura 14.c). En caso de seleccionar el control remoto, despliega su componente, en caso de seleccionar otro comportamiento se mostrará el componente de “Custom-Behaviour”.

ImageLoader: Muestra imágenes/gifs en el centro de la pantalla cuando los comportamientos lo requieren (figura 14.a). Cuenta con una colección de imágenes/gifs entre las cuales el comportamiento puede elegir una.

Componentes adicionales: Se dispone además de componentes que despliegan mensajes informativos al usuario. Por ejemplo, se informa cuando la aplicación está cargando (figura 14.d), o en caso de que el dispositivo no esté conectado al Wifi de Robotito, o cuando la base de datos local se encuentra vacía y es necesario conectarse a internet para descargar los comportamientos de la base remota.

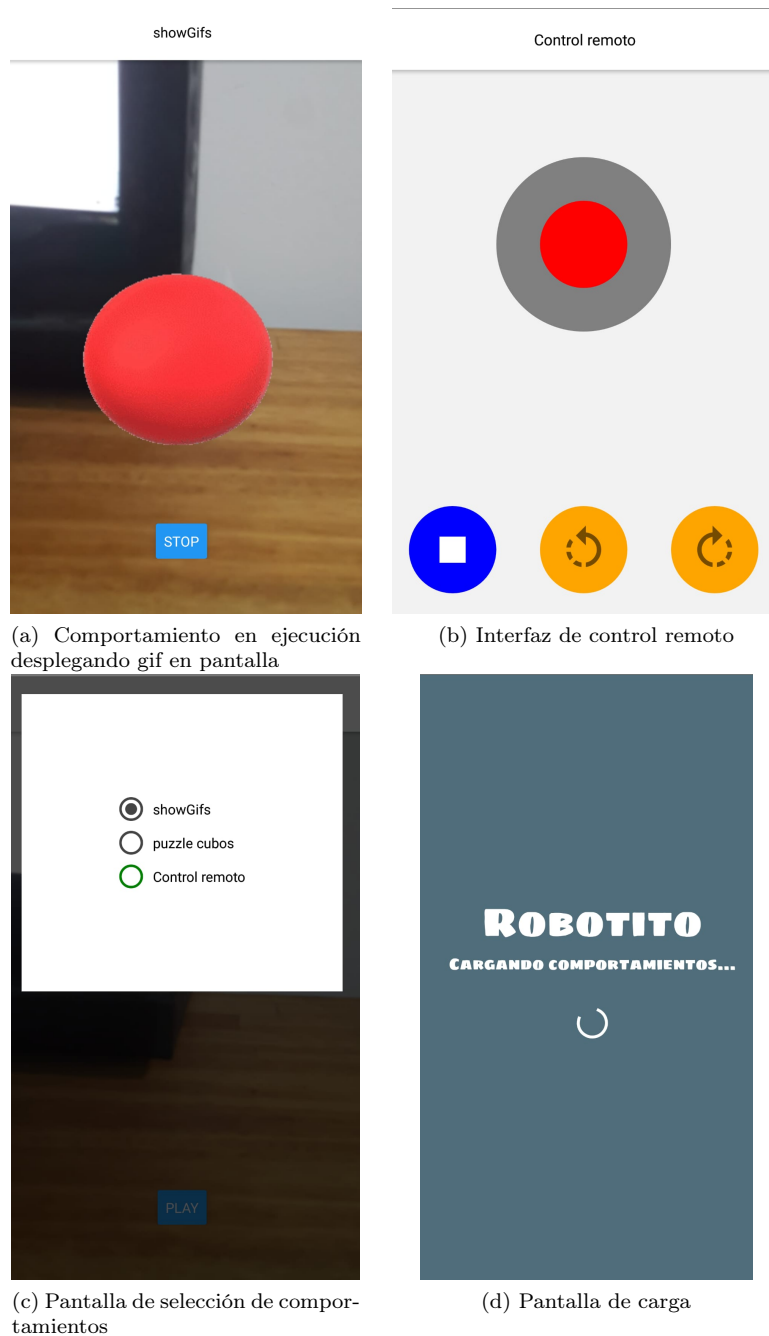


Figura 14: Interfaz gráfica de la aplicación

Capa lógica La capa lógica agrupa todos aquellos componentes que implementan las reglas sobre cómo se comporta Robotito, sobre cómo se procesan las entradas recibidas desde la capa de presentación y cómo se utilizan los datos obtenidos desde la capa de datos.

App: Es el punto de entrada de la aplicación, inicializa la capa de presentación (al iniciar el componente “BehaviourSelector”), obtiene la definición de los comportamientos almacenados a través de la capa de datos, e inicializa el *concurrent state machine controller* (CSMController).

CSMController: Es el encargado de manipular la máquina de estados jerárquica a través del CSMMotor, utilizando las especificidades del sistema. Es quien le dice a la máquina de estados qué recursos utilizar (pantalla, sonido y los motores de Robotito), qué definición de comportamiento cargar, y cuándo comenzar a ejecutarlo o detenerlo. Al inicializarse desde App, carga una lista de definiciones de comportamientos, y cuando un usuario selecciona un comportamiento a través de la interfaz, el CSMController detiene la ejecución de la máquina de estados y la vuelve a inicializar con la definición del comportamiento seleccionado. También recibe los eventos desde el ImageProcessor y los propaga al motor de la máquina de estados.

ImageProcessor: Este componente recibe frames de la preview de la cámara, los procesa utilizando modelos de reconocimiento de imágenes, y según lo detectado en cada frame se envían eventos al CSMController. Los modelos utilizados para reconocimiento de imágenes son, “Barcode Scanning” de MLKit, que resuelve la detección de marcas; Image Labeler también de MLKit, que permite etiquetar imágenes según la probabilidad de que algunos objetos genéricos se encuentren en ella. El tercer modelo utilizado se encarga de detectar formas geométricas y es de elaboración propia, entrenado utilizando Google Cloud. Este módulo se desarrolló en Android Native debido a que es más eficiente procesar datos a más bajo nivel, que enviarlos a React Native para procesarlos.

Sound/Screen Controller: Estos dos componentes implementan las funciones que se utilizan en el motor de máquina de estados cuando se utilizan como recursos. En el caso de ScreenController, avisa al Image Loader cuando debe desplegar una imagen a través del uso del patrón de diseño “Observer”. En el caso de SoundController, directamente reproduce el sonido.

RobotitoInterface: Implementa la conexión con Robotito. La misma se hace mediante un socket udp y el envío a través de mensajes por broadcast. Los mensajes se envían cada 100 milisegundos debido a que es necesario para que Robotito se mantenga escuchando comandos. El comando que se le envía es:

$$speed * velX * velY * velZ1 * velZ2$$

Donde $velX$, $velY$ y $velZ1$ son las velocidades en los ejes X, Y y de rotación respectivamente, el parámetro $velZ2$ también puede ser utilizado pero Robotito no hace nada con él, pero dado que se encontró en la documentación se decidió incluirlo pensando en futuros desarrollos (por ejemplo, se podría utilizar para un motor que gire o incline el dispositivo móvil).

CSMMotor: Se desarrolló este componente en React Native, el cual implementa una máquina de estados jerárquica. Se inicializa al recibir los recursos a utilizar y la definición de las máquinas de estados que se desea ejecutar. Al inicializarse, se guarda el estado inicial de cada máquina de estado como estados actuales. Cada vez que carga un estado como estado actual, revisa sus transiciones para prepararlas en caso de que sean por tiempo. Además, el motor cuando recibe eventos externos, revisa las transiciones de sus estados actuales, si alguna de esas transiciones es por evento y evalúa si corresponde cambiar de estado. El motor puede detenerse o reanudarse mediante sus funciones `RunCSM` y `StopCSM`.

Capa de datos La capa de datos cuenta de un solo componente, es la encargada de realizar la conexión con la base de datos local y la remota. Los datos que se almacenan son las definiciones de los comportamientos.

DataBaseController: Ambas bases de datos son implementadas por el servicio “realtime database” de Firebase mencionado en el análisis de tecnologías. Este controlador utiliza su biblioteca “react-native-firebase/database” para obtener todas las definiciones de comportamientos almacenadas en la base de datos remota (en caso de tener conexión a internet). A su vez, esos datos son persistidos por la biblioteca en la base de datos local para que luego se pueda trabajar sin conexión. Además, el componente se encarga de detectar a través de un evento que nos provee la biblioteca cambios en la base de datos remota.

4.1.2. Comportamientos

Una parte central de la aplicación y del proyecto son los comportamientos, que definen el conjunto de reglas que va a gobernar las acciones de Robotito y cómo reacciona a su entorno.

Máquinas de estados concurrentes Se encargan de determinar cómo actúa el robot y qué gifs o sonidos se reproducen en cada momento en base a los elementos que se detectan en su entorno.

Debido a que se desea contar con una colección de comportamientos que puedan ser cargados, puestos en marcha o pausados, es necesario contar con un motor de máquinas de estados concurrentes, que durante la ejecución de la aplicación sea capaz de leer las definiciones de los comportamientos y ejecutar las máquinas de estados que los representan.

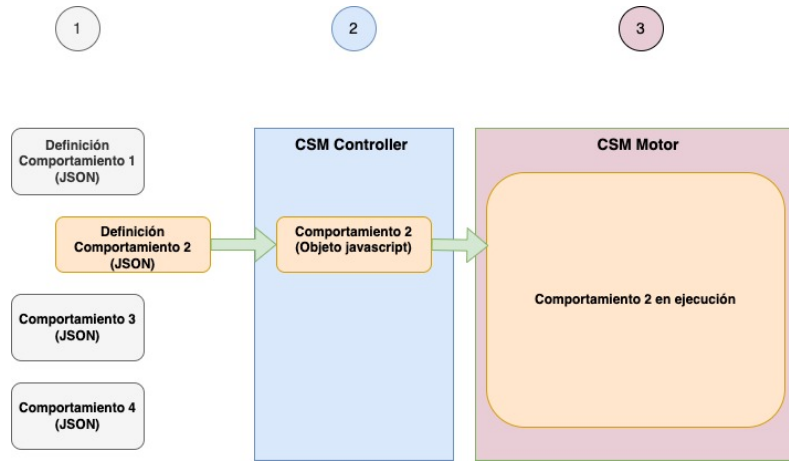


Figura 15: Flujo de los comportamientos: definición, cargado y ejecución.

En primer lugar, la base de datos almacena una lista de definiciones de comportamientos en formato JSON (punto 1, figura 15). Esto tiene varias ventajas, ya que es un formato sumamente flexible que permitiría a investigadores agregar cambios para nuevas funcionalidades fácilmente. Además, es un formato de intercambio de datos muy utilizado para webs y aplicaciones, en particular en aquellas desarrolladas en frameworks de Javascript por la similitud semántica entre JSON y los objetos de Javascript. Teniendo en cuenta que React Native utiliza Javascript, se decidió por JSON. Otro motivo para esta elección es que la tecnología utilizada para la base de datos (Realtime Database) trabaja con JSON.

Segundo, el CSMController carga la lista de comportamientos en memoria y cuando se selecciona uno, se lo pasa al motor quien lo comenzará a ejecutar (punto 2, figura 15). Otras tareas que cumple el CSMController son inicializar el motor, indicarle cuáles son los recursos que se utilizarán, pasarle los eventos que se reciben desde el procesador de imágenes, la interfaz, o desde el comportamiento en ejecución, e indicarle al motor cuando pausar o reanudar los comportamientos.

CSMMotor es capaz de a partir de un objeto Javascript levantar máquinas de estados concurrentes (punto 3, figura 15), es quien lleva registro de los estados actuales del comportamiento, se fija en los recursos que utilizan, y evalúa cuándo se debe pasar de un estado a otro. Además, puede pausar o reanudar la máquina de estados, y recibe eventos que pueden desencadenar transiciones en sus estados actuales.

Definición de un comportamiento Según las funcionalidades propuestas, un comportamiento debe poder ser almacenado en forma de algún tipo de registro y que ese registro pueda ser transmitido entre la base de datos remota y la aplicación. Dadas esas condiciones se eligió definir y almacenar los com-

portamientos en formato JSON. Se detalla a continuación (figura 16) cómo se modelaron las definiciones de comportamientos JSON.

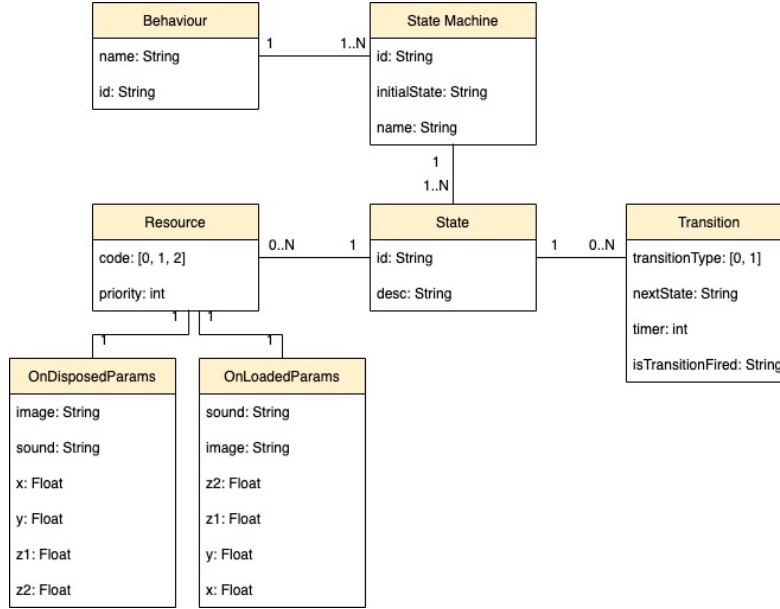


Figura 16: Estructura de la definición de un comportamiento

Cada comportamiento se compone de varias máquinas de estados, las cuales tienen una colección de estados y especifican el estado inicial. Cada estado representa una acción de Robotito, por lo que contiene una colección de recursos que está utilizando para esa acción y un conjunto de condiciones para cambiar de estado y realizar otras acciones.

Los recursos tienen un código que los identifica: 0 representa los motores omnidireccionales, 1 el sonido del dispositivo y 2 la reproducción de gif o imágenes en pantalla. Además cuentan con una prioridad; en caso de que varios estados intenten usar el mismo recurso a la vez, lo utiliza aquel con la prioridad más alta. Finalmente, los recursos tienen los parámetros a usar al ser cargados o al liberarlos (OnLoadedParams y OnDisposedParams). Al ser cargado el recurso “ruedas omnidireccionales” los parámetros son las velocidades en cada eje. En el caso del recurso “pantalla”, el parámetro es el nombre de la imagen o gif a ser mostrado. Mientras que al cargar un sonido se requiere el nombre. En el caso de las transiciones, se utilizan dos tipos: aquellas que se ejecutan por tiempo (transitionType 0), que llevan un timer en milisegundos, y aquellas que se ejecutan por eventos (transitionType 1). Las transiciones basadas en eventos tienen un campo “isTransitionFired” que es el cuerpo de una función en formato de texto. Esta función recibe el evento como parámetro “event”, y devuelve True en caso de que el evento provoque un cambio de estado, y False en caso contrario.

Ejemplo de definición de comportamiento

```
{id: "esferaRojaGiro", name: "esfera roja y girar",
stateMachines:[

  {id: "esferaRojaGiroSM",
  name: "esfera roja y girar",
  initialState: "esperarEsferaRoja",
  states: [
    {id: "esperarEsferaRoja",
    desc: "Esperar por la esfera roja",
    transitions: [
      {transitionType: 1,
      isTransitionFired: "if (event.geometricLabels) {var toReturn = false; event.geometricLabels.forEach(function(label) { if (label.text == 'esfera-rojo' && label.confidence >0.55) { toReturn = true; } }); return toReturn; } else { return false; } ",
      nextState: "girar1segundo"}}]
    },
    {id: "girar1segundo",
    desc: "Girar por un segundo",
    transitions: [
      {transitionType: 0,
      timer: 1000,
      nextState: "esperarEsferaRoja"}],
    resources: [
      {code: 2,
      onDisposedParams: { image: "" },
      onLoadedParams: { image: "RedBall" },
      priority:10
      },
      {code: 0,
      onDisposedParams: {z1: 0},
      onLoadedParams: {z1: 1.5},
      priority:10 }
    ]
  }
]}
}
```

El comportamiento definido genera la máquina de estados de la figura 17:



Figura 17: Máquina de estados del comportamiento de ejemplo.

Cuando ejecute este comportamiento, Robotito esperará hasta que detecte una esfera roja, en ese momento girará sobre su eje por un segundo y mostrará un gif de una esfera roja en pantalla. Luego volverá a esperar a detectar una esfera roja.

Ejecución de un comportamiento La figura 18 muestra los módulos involucrados en la ejecución de un comportamiento. Para ejecutarlos, el usuario selecciona uno del menú de selección de comportamientos. Luego el CSMController busca la definición del comportamiento seleccionado entre la lista de comportamientos que tiene en memoria. Revisa en el comportamiento los campos “isTransitionFired” de las transiciones por eventos y los convierte a funciones. En este punto, el comportamiento es un objeto Javascript. Finalmente, el CSMController llama a la función startCSM del CSMMotor con la definición del comportamiento (objeto Javascript).

Una vez se llama a startCSM en el motor de máquinas de estados, se descarta cualquier ejecución en curso que tuviera. Luego, para cada máquina de estados del comportamiento, se carga el estado inicial en la lista de estados actuales. Para cada estado actual se revisan sus recursos, y se registra para cada recurso el estado actual con mayor prioridad sobre él. El comportamiento se carga en pausa y se espera que desde la interfaz se presione “play”.

Cuando desde la interfaz se le da iniciar al comportamiento, el CSMController ejecuta la función runCSM del CSMMotor, la cual, para los estados actuales hace dos cosas: crea timeouts para las transiciones por tiempo según sus timer. Y en caso de que el estado tenga prioridad sobre algún recurso, se ejecuta la función onLoading del recurso con los parámetros obtenidos de onLoadingParams en el estado. Esto provoca que se preparen los cambios de estado en caso de transiciones por tiempo y se comiencen a utilizar los recursos que corresponden con los parámetros establecidos en los estados. Finalmente se setea el motor como “running”.

En el caso de parar un comportamiento desde la interfaz, el CSMController llama a la función stopCSM del motor, la cual borra los timeouts de las tran-

siciones por tiempo y llama a las funciones onDisposed de los recursos, con los parámetros obtenidos de onDisposedParams definidos en el estado con prioridad sobre ese recurso. Además, se cambia la propiedad running del motor a false.

Los eventos son enviados desde el ImageProcessor al CSMController, quien ajusta levemente el formato del evento y llama a la función fireEvent del CSM-Motor. El motor revisa para cada estado actual si tiene transiciones por evento, en cuyo caso, si la función isTransitionFired con el evento como parámetro evalúa en verdadero, se cambia de estado para el siguiente estado de la transición.

Cada vez que el motor cambia sus estados actuales, sea por transiciones por evento o por transiciones por tiempo, se hacen las siguientes tareas:

- Se eliminan los timeouts de las transiciones del estado anterior.
- Se saca el estado anterior de la lista de estados actuales.
- Se carga el nuevo estado en la lista de estados actuales.
- Se vuelve a detectar los estados con más prioridad sobre cada recurso por si cambiaron, y se cargan o liberan los recursos que correspondan.

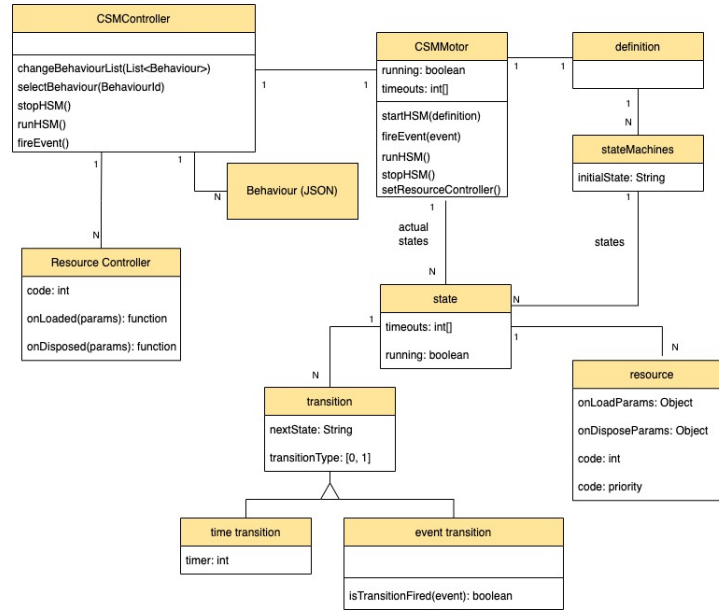


Figura 18: Diagrama de entidades que participan de la ejecución de un comportamiento

4.1.3. Reconocimiento de imágenes, marcas

Para el reconocimiento de códigos se utilizó barcode-scanning de MLkit [33] versión 16.1.1, que permite detectar en una imagen la posición y los datos codificados de los códigos que aparezcan. No necesita conexión a internet para su

funcionamiento y puede reconocer hasta diez códigos en la misma imagen. En particular, puede detectar:

- Formatos lineales: Codabar, Code 39, Code 93, Code 128, EAN-8, EAN-13, ITF, UPC-A, UPC-E
- Formatos en 2D: Aztec, Data Matrix, PDF417, QR Code

Para el presente proyecto, se utilizaron solamente las marcas en formato Aztec y QR Code, pero el resto de los mencionados pueden ser fácilmente habilitados.

El modelo puede trabajar con imágenes de diversos tamaños, pero cuanto mayor sea la imagen, mayor será la latencia. En contrapartida, según la documentación, la marca a escanear tiene que tener en la imagen un mínimo de 2 píxeles de ancho y 2 píxeles de alto. Debido a que la aplicación es en tiempo real, y la latencia es importante, se decidió trabajar con imágenes más chicas de lo recomendado para la mayoría de las aplicaciones. Esto implica que la marca a escanear debe estar cerca de la cámara para alcanzar el mínimo de píxeles soportado. El tamaño de imagen utilizado es el más chico de los soportados por el celular, que tenga al menos 480 px de ancho y 360 px de alto. Como entrada del modelo se utilizó un `InputImage`, creado a partir de la preview de la cámara obtenida en formato NV21 [34].

Luego de procesada la imagen, el modelo devuelve una lista de las marcas detectadas, donde para cada marca se dispone de: el rectángulo que lo contiene; las coordenadas de las cuatro esquinas; el valor codificado y el tipo de información codificada.

4.1.4. Reconocimiento de imágenes, general

Para esta funcionalidad se usó la librería `image-labeling`[35], también de `ML-Kit`, en su versión 17.0.3. Ésta cuenta con una lista de 400 elementos u objetos que es capaz de detectar si están presentes en una foto. La lista de elementos es variada, y en general son de elementos cotidianos, por ejemplo: mano, auto o lentes.

Al igual que para el caso del reconocimiento de marcas, se utilizó como entrada del modelo imágenes del tamaño más chico soportado por la cámara que tenga al menos 480 px de ancho y 360 px de alto. De la misma forma que para el modelo de reconocimiento de marcas la entrada del modelo es un `InputImage`[36], creado a partir de la preview de la cámara.

Luego de procesada la imagen, el modelo retorna para cada etiqueta su índice, su nombre y la confianza de que el elemento se encuentre en la imagen.

4.1.5. Reconocimiento de Figuras geométricas

Para implementar la funcionalidad de etiquetado de imágenes se decidió implementar un modelo desde cero. Si bien implica un mayor trabajo, ya que es necesario elaborar el conjunto de datos para entrenarlo, nos permite explorar cómo se desarrollan nuevos modelos para resolver problemas específicos. Poder elaborar modelos a medida e integrarlos a la aplicación favorece su flexibilidad.

Construcción del dataset El primer paso para la creación del modelo fue elaborar el dataset (conjunto de datos), para lo que se construyeron figuras geométricas de colores, con cartón y papel glacé (figura 29). Se eligieron esos materiales porque son comunes en cualquier escuela o liceo. Las formas construidas tienen los siguientes tamaños:

Cubo: 5.1cm x 5.1cm x 5.1cm

Pirámide: base cuadrada de 5.8cm x 5.8cm, altura de 5.8cm

Esfera: 5 cm de diámetro



Figura 19: Figuras geométricas construidas

Luego, para cada figura se tomaron 100 fotos (ejemplos en la figura 20), variando el ángulo, la distancia a la cámara, la luz y el fondo. Se tomaron 100 fotos más sin ninguna figura, ya que interesa detectar también cuando no hay ninguna presente. Luego se utilizó la herramienta Vision de Google Cloud para elaborar el conjunto de datos de tipo “clasificación con una sola etiqueta”. Se agregaron las fotos tomadas y se etiquetó cada imagen en una categoría según qué forma geométrica aparece en ella.

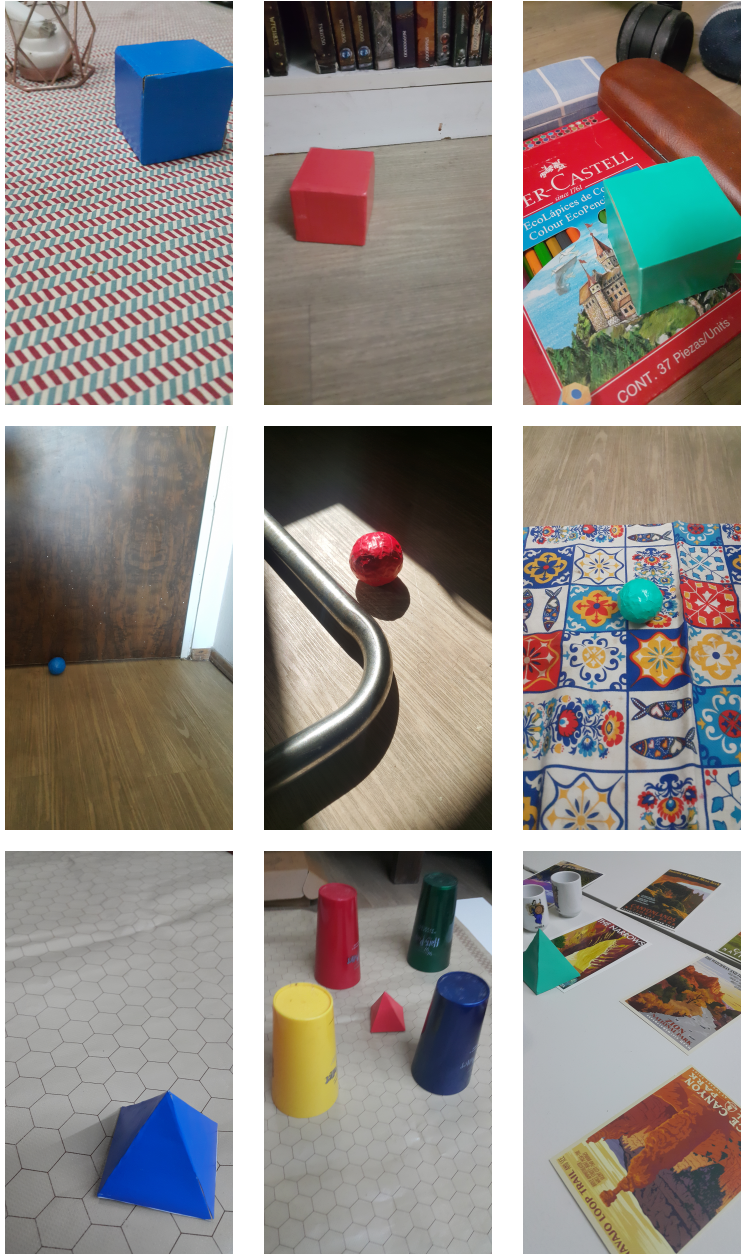


Figura 20: Ejemplo de fotos con las que se entrenó el modelo.

De las 1000 fotos utilizadas, 900 se utilizaron como conjunto de entrenamiento y 100 como conjunto de test. La herramienta utilizada solo permitió dividir el conjunto de datos en esa proporción, pero se considera una proporción

aceptable para el tamaño del dataset. Al evaluarlo, se encontró que el modelo tiene una precisión promedio de 94.8 %. Las siguientes gráficas muestran cómo se comporta el modelo en función de la confianza seleccionada, ya que la misma es un parámetro que puede ser utilizado en tiempos de ejecución. Se considera que, para el uso que se le va a dar al modelo, es prioritario contar con una buena precisión sobre una buena recuperación, ya que es preferible no detectar alguna figura en un frame y confiar en que se detectará en el siguiente frame, a confundir una figura con otra, lo cual alteraría el comportamiento de Robotito de una forma inesperada para los usuarios. Mirando la gráfica de recuperación y precisión (figura 21) es interesante observar que incluso para niveles de confianza de 0.22 la precisión sigue manteniéndose por encima de 90 %. Las pruebas de funcionalidad de la sección de experimentación se realizaron utilizando un nivel de confianza de 0.7, lo que nos permite tener una precisión de 98.7 % con una recuperación de 76 %.

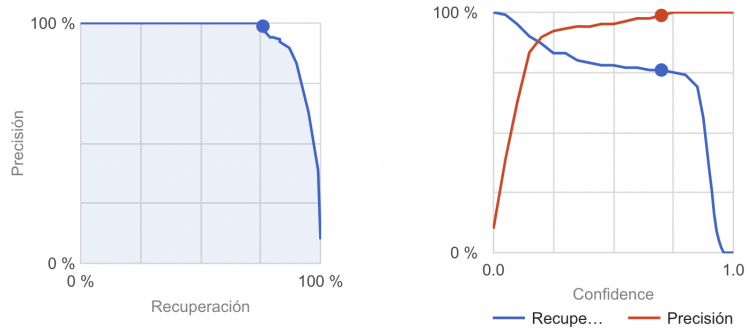


Figura 21: Recuperación en función de la precisión (izquierda), confianza en función de recuperación y precisión (derecha). El punto azul y rojo muestran las gráficas al nivel de confianza de 0.7.

Luego se analizó la matriz de confusión:

Etiqueta de confianza	Etiqueta predicha									
	esfera-verde	cubo-azul	cubo-verde	piramide-azul	piramide-roja	esfera-azul	cubo-rojo	esfera-roja	piramide-verde	otros
esfera-verde	100 %	-	-	-	-	-	-	-	-	-
cubo-azul	-	100 %	-	-	-	-	-	-	-	-
cubo-verde	-	-	80 %	-	-	-	-	-	10 %	10 %
piramide-azul	-	-	-	100 %	-	-	-	-	-	-
piramide-roja	-	-	-	-	100 %	-	-	-	-	-
esfera-azul	-	-	-	-	-	100 %	-	-	-	-
cubo-rojo	-	-	10 %	-	30 %	-	50 %	-	-	10 %
esfera-roja	-	-	-	-	-	-	-	90 %	-	10 %
piramide-verde	-	-	-	-	-	-	-	-	100 %	-
otros	10 %	10 %	-	-	10 %	10 %	-	-	-	60 %

Figura 22: Matriz de confusión para las figuras geométricas.

Algunas conclusiones que surgieron de la matriz de confusión (figura 22), fueron que el modelo tiene algunas dificultades para discernir entre el cubo rojo y la pirámide roja. Es posible que sea debido a que la pirámide al ser de base cuadrada, se vea igual al cubo desde cierto ángulo. Esto podría ser evitado con pirámides de base triangular (tetraedros). En la etiqueta “otros” solamente se acierta un 60 % de las veces. Esta fue pensada para que el modelo la utilice cuando no hay figuras en la imagen. Debido a su bajo porcentaje de acierto, se considera que una estrategia podría ser eliminar la etiqueta y considerar que no hay figuras cuando las probabilidades de las demás etiquetas son bajas. De todas formas, en el resto de las figuras no tiene mayores problemas el modelo, y la matriz da una buena idea de con cuáles figuras conviene trabajar para que el modelo funcione mejor.

5. Experimentación

Con el fin de examinar el comportamiento de la aplicación, se ejecutó un caso de uso que abarca todas las funcionalidades. Además, se tomaron mediciones de la aplicación para evaluar su funcionamiento. En particular se midió:

- Tiempo en iniciar un comportamiento, medida que permite ver el desempeño del motor de máquina de estados y de la aplicación en general.
- Velocidad máxima del robot utilizado, ya que para crear comportamientos es relevante tener una idea general de los límites físicos del robot.
- Velocidad de captura de frames de la cámara. Para una aplicación en tiempo real que utiliza principalmente la cámara es importante poder capturar frames de la cámara a una buena velocidad.
- Tiempo de procesamiento de cada modelo de reconocimiento de imágenes. Esta medida permite evaluar los distintos modelos utilizados, cuanto menor tiempo de procesamiento tienen mejor será la experiencia de usuario, ya que el robot reaccionará más rápido a su entorno.
- Tiempo de respuesta usando el reconocimiento de figuras geométricas. Es una buena aproximación al tiempo de respuesta que percibe el usuario al utilizar la aplicación con este modelo, desde que el usuario le muestra una figura geométrica al robot hasta que la aplicación toma una acción basándose en eso.

Al ser una aplicación en tiempo real, es relevante estudiar el tiempo que toma realizar estas acciones y mantenerlos lo más bajo posible.

Para realizar las pruebas se utilizó un celular Samsung Galaxy Note 4 [38], con sistema operativo Android 6.0.1. La resolución de su pantalla es de 2560 x 1440 (Quad HD) y su cámara cuenta con una resolución de CMOS 16.0 MP. En cuanto a los videos, graba en una resolución de UHD 4K (3840 x 2160) a 30 frames por segundo.

5.1. Pruebas de funcionalidades

Para validar el funcionamiento de los comportamientos y el etiquetado de figuras geométricas, se definió un comportamiento que al detectar ciertas figuras geométricas, desencadena un patrón de movimiento. Luego, se marcó una pista en el suelo, y la prueba implicó ir moviendo al robot mostrándole las figuras adecuadas, siguiendo la pista sin salirse de la misma.

Los patrones de movimientos cargados fueron los siguientes:

- El cubo rojo provoca que el robot gire en sentido antihorario 90°.
- Al detectar el cubo azul, el robot avanza una casilla, gira en sentido horario 90° y luego avanza otra casilla.

- El cubo verde hace que el robot gire 90° en sentido horario y avanza tres casillas al frente.

El comportamiento definido puede ser representado por la máquina de estados de la figura 23.

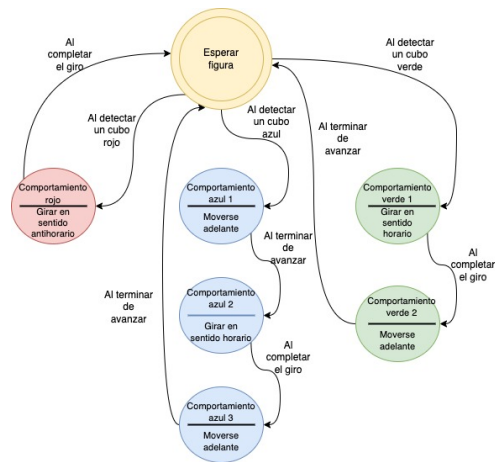


Figura 23: Máquina de estados para el comportamiento de prueba

Notar que tanto los grados de giro como las casillas, se establecieron eligiendo experimentalmente la velocidad y el tiempo de movimiento adecuados.

La pista marcada para la prueba tenía la forma mostrada en la figura 24:

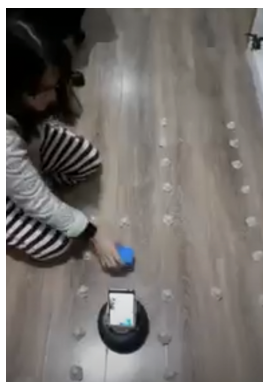


Figura 24: Forma y dimensiones de la pista

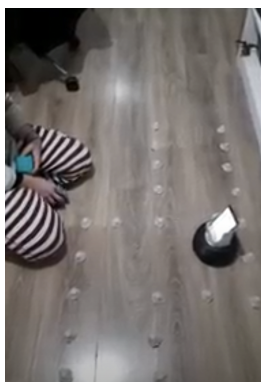
Se realizó la prueba con la participación de una persona externa al proyecto, quien no participó en la definición del comportamiento de prueba y desconocía el

mismo. Al usuario se le permitió probar libremente cómo reaccionaba Robotito a las distintas figuras durante unos minutos, dándole una mínima introducción de que Robotito se movería de distintas maneras ante cada figura. Luego debía resolver la tarea de llevar a Robotito al final de la pista.

La primera figura que se utilizó fue el cubo azul (figura 25.a), por lo que el robot avanzó, giró 90° y volvió a avanzar (figura 25.b). La siguiente figura que se utilizó fue el cubo rojo (figura 25.c), provocando que el robot girara en sentido antihorario (figura 25.d). Al mostrarle el cubo rojo dos veces, se logra un giro de 180° (figuras 25.e y 25.f). Finalmente, se le muestra el cubo verde al robot (figura 25.g), logrando así que gire 90° en sentido horario y avance hasta el final de la pista (figura 25.h).



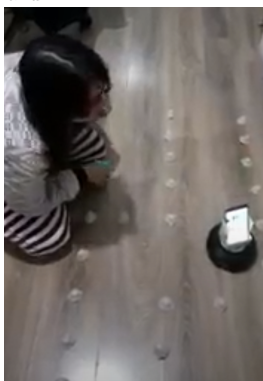
(a) Se muestra el cubo azul



(b) Posición luego del cubo azul



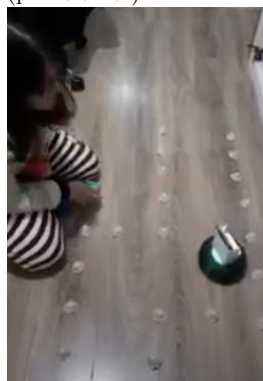
(c) Se muestra el cubo rojo (primera vez)



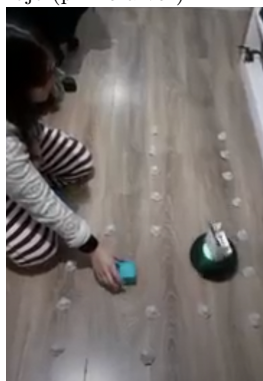
(d) Posición luego del cubo rojo (primera vez)



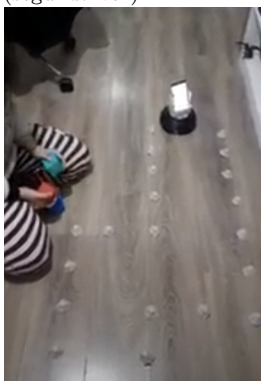
(e) Se muestra el cubo rojo (segunda vez)



(f) Posición luego del cubo rojo (segunda vez)



(g) Se muestra el cubo verde



(h) Posición luego del cubo verde

Figura 25: Prueba de funcionalidad, reconocimiento de figuras geométricas

5.2. Mediciones de funcionamiento

Tiempo en iniciar un comportamiento El primero de los tiempos que se calculó fue el tiempo entre que se inicia un comportamiento desde la interfaz y que Robotito recibe el comando de moverse, suponiendo que lo primero que hace el comportamiento es mover al robot. Este tiempo nos da una idea de cuanto se demora en comenzar a ejecutar un comportamiento y en enviar un comando de movimiento a través de la interfaz de Robotito. Algo a considerar es que, debido a que para controlar el movimiento del robot se envían mensajes cada 30 milisegundos, es posible que la mayor demora se produzca esperando a que se envíe el siguiente mensaje.

Para esta prueba se tomaron 10 mediciones:

Tiempo en iniciar
27 ms
19 ms
39 ms
32 ms
30 ms
29 ms
30 ms
50 ms
35 ms
33 ms

El tiempo promedio en iniciar fue de 33 ms, mientras que el peor tiempo fue de 50 ms. La desviación estándar es 7.70. La variación en los tiempos de iniciar se debe probablemente a otros procesos corriendo en el celular en paralelo de cosas no relacionadas, y a lo explicado anteriormente de que los mensajes al robot se envían cada 30 milisegundos, por lo que una orden de movimiento que llegue próxima al siguiente mensaje va a demorar menos que una que llegue apenas se mandó el mensaje anterior.

Velocidad del robot Para obtener la velocidad máxima del Robotito utilizado, se preparó un comportamiento en el cual el robot se moviera para adelante a su velocidad máxima (0.8m/s). Luego se marcó en el piso una distancia de 2 metros, y se tomó el tiempo que tardó el robot en recorrerlos, se repitió la prueba 9 veces.

Tiempo en recorrer 2 metros
2840 ms
2910 ms
2980 ms
2870 ms
2900 ms
2820 ms
2850 ms
2870 ms
2950 ms

A Robotito le tomó en promedio 2888 milisegundos recorrer los 2 metros, de lo que se deduce una velocidad de 0.69 m/s como máximo en la práctica, para el Robotito examinado en las condiciones de las pruebas. La velocidad máxima teórica estipulada en el firmware de Robotito es de 0.8 m/s. La diferencia con la velocidad obtenida en la práctica se puede dar debido a falta de batería, desperfectos en los motores o en las ruedas, al rozamiento de sus ruedas, el paso extra del celular, o otras características del robot particular que se está utilizando o el entorno donde se lo está probando. En la prueba que tardó más demoró 2980 ms en hacer los 2 metros. La desviación estándar fue de 52.39.

Tiempo de captura de frames de la cámara Se tomó la medida de cuántos frames por segundo son capturados por la cámara. Es de relevancia porque en caso de que se capturen pocos frames por segundo, implica una limitante a la velocidad a la que Robotito reacciona a cambios en su entorno. Si Robotito reacciona lento, dificulta que sus usuarios logren asociar de qué forma cambios en el entorno modifican el comportamiento del robot. Para tomar esta medición se incrementó un contador cada vez que se capturaba un frame de la cámara, y se ejecutó la aplicación por 59.8 segundos.

Se capturaron 998 frames durante la ejecución (que duró 59.8 segundos), o sea 16.6 frames por segundo. Se considera una cantidad aceptable de frames para el dispositivo en el que se lo probó, que graba en un framerate de 30 frames por segundo.

Tiempo de procesamiento de los modelos La siguiente prueba que se realizó fue obtener el tiempo en milisegundos que se tarda en procesar la imagen utilizando cada uno de los modelos utilizados. Para esto se tomó el tiempo en milisegundos al inicio y final del procesamiento, se repitió el experimento 10 veces. Los tiempos obtenidos son:

Códigos QR	Image labeler	Figuras geométricas
26 ms	159 ms	430 ms
78 ms	164 ms	394 ms
48 ms	183 ms	393 ms
82 ms	155 ms	421 ms
55 ms	182 ms	360 ms
67 ms	139 ms	395 ms
74 ms	180 ms	350 ms
40 ms	133 ms	398 ms
40 ms	189 ms	414 ms
69 ms	173 ms	407 ms

El tiempo promedio para procesar códigos QR es de 57.9 ms, lo que es equivalente a 17,27 frames por segundo. Para el caso del modelo Image Labeler, el promedio es 165.7 ms, o 6.04 frames por segundo. Finalmente, el modelo de reconocimiento de figuras geométricas demora en promedio 396.2 ms, o lo que es lo mismo, 2.52 frames por segundo. Para evaluar estos resultados hay que considerar que los tres modelos se procesan en paralelo, y se obtuvo, como consecuencia, que los tiempos de cada modelo no se suman al calcular el tiempo que se tarda en procesar los tres. Además, dejar de procesar uno de los modelos podría disminuir el tiempo de procesamiento de los otros.

Tiempo de respuesta Para analizar el tiempo de respuesta, se implementó un comportamiento que utilizara el modelo de figuras geométricas (el modelo que tarda más en procesar), y que al detectar un cubo rojo despliegue un gif en pantalla. Para este comportamiento se midió el tiempo entre que se le da inicio y se despliega el gif en pantalla. Para este experimento se descartaron los frames que ya se estaban procesando antes de comenzar el comportamiento.

Tiempo de respuesta (milisegundos)
464 ms
435 ms
408 ms
374 ms
210 ms
331 ms
290 ms
325 ms
285 ms
452 ms

El tiempo promedio de respuesta es 357.4 milisegundos (2.8 frames por segundo). El peor tiempo obtenido es 464 milisegundos, y la desviación estándar 83.3. Esta prueba da una medida de cuánto tarda Robotito en reaccionar a un

cambio de su entorno utilizando el más lento de sus modelos. Experimentalmente se observa que este tiempo es aceptable para que los usuarios puedan asociar qué cambios del entorno desencadenan qué comportamientos del robot, y para que el uso de la aplicación sea fluido.

6. Conclusiones y trabajo a futuro

El proyecto cumplió con los objetivos planteados en un inicio, ya que permite expandir las capacidades del robot mediante un dispositivo móvil utilizando la cámara para proporcionarle visión al robot mediante reconocimiento de imágenes, además de utilizar su pantalla y parlantes como nuevas formas de interacción. Se considera que la forma en la que se implementó prioriza la flexibilidad, permitiendo a los usuarios desarrolladores construir nuevos comportamientos y subirlos a una base de datos para que todos los que tengan la aplicación puedan usarlos.

Se utilizó reconocimiento de imágenes para mejorar la interacción con el entorno de tres formas distintas: reconocimiento de marcas, etiquetado de imágenes con objetos generales y etiquetado de formas geométricas.

Es interesante que el modelo y el conjunto de datos de las formas geométricas fue elaborado desde cero, por lo que siguiendo la misma metodología se podrían elaborar otros modelos que tengan mayor interés para los usuarios. Su funcionamiento fue correcto, ya que con un nivel de confianza de 0.7, se obtuvo una precisión de 98.7 % y una recuperación de 76 %, lo cual se considera aceptable.

Los tiempos de ejecución de los modelos utilizados durante la experimentación fueron aceptables y la aplicación reacciona rápidamente a los cambios en su entorno. De todas formas, es posible optimizar más el modelo creado para las figuras geométricas, para mejorar la rapidez del reconocimiento. Algunas ideas de cómo optimizarlo podrían ser trabajar con imágenes más chicas, menos nodos, o menos categorías (por ejemplo, eliminar “otros” que no resultó de utilidad) ya que simplificaría la red neuronal y el costo de procesamiento.

Sin embargo, el etiquetado de imágenes (general) no funcionó de la forma esperada. La lista de elementos que detecta es muy grande y genérica, por lo que a veces al mostrarle un “brazo” detecta “piel”, o no detecta consistentemente algunos elementos. A pesar de eso algunos elementos son detectados correctamente y pueden ser utilizados fácilmente en la aplicación (por ejemplo, “mano”).

Además de la cámara, se agregaron la pantalla y el sonido del dispositivo móvil como formas de interacción. Si bien lo implementado permite desplegar gifs y reproducir algunos sonidos, se considera que en este punto se puede mejorar. La interfaz gráfica podría incluir animaciones o tener un diseño más atractivo. Se podrían extender las colecciones de sonidos y los gifs para utilizar, incluyendo elementos que estén relacionados con dinámicas que se piensen realizar en el aula.

Sobre la base de datos utilizada, es una buena solución dado que permite sincronizar una base de datos en la nube con bases de datos locales en cada dispositivo, y que al utilizar el formato JSON, que es un estándar, permite ser integrada fácilmente con futuros proyectos. Sin embargo, se detecta como oportunidad de mejora que se permita configurar bases de datos por escuelas o grupos, de forma que no todos compartan la misma base de datos. Además, otro trabajo a futuro que podría implementarse es una web que permita crear los comportamientos y los suba a la base de datos, o la elaboración de repositorios de comportamientos, un IDE que facilite la creación de los mismos y la

posibilidad de compartirlos por redes sociales. Sería interesante que funcione de forma similar a algunos lenguajes gráficos como Scratch [39], de forma que los alumnos puedan participar de la creación de comportamientos.

Finalmente, se considera importante probar el funcionamiento del robot con el dispositivo móvil y la aplicación creada en este proyecto en el aula con los usuarios reales, ya que para las siguientes iteraciones del proyecto sería bueno contar con opiniones y experiencias de los usuarios.

Referencias

- [1] Robótica Educativa De México S.A. (2009). Concepto.
<https://www.roboticaacademica.com/quienes-somos3>
- [2] Talentiam (2019). Learning by doing, la metodología orientada a la acción.
<https://www.talentiam.com/es/blog/learning-by-doing-metodologia-accion>
- [3] Tzagkaraki, E., Papadakis, S., & Kalogiannakis, M. (2021, February). Exploring the Use of Educational Robotics in primary school and its possible place in the curricula. In Educational Robotics International Conference (pp. 216-229). Springer, Cham.Exploring the Use of Educational Robotics in Primary School and Its Possible Place in the Curricula, 2019.
- [4] Bakala, E., Visca, J., Tejera, G., Seré, A., Amorin, G., & Gómez-Sena, L. (2019, October). Designing child-robot interaction with Robotito. In 2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN) (pp. 1-6). IEEE.
- [5] Tejera, G., Amorin, G., Sere, A., Capricho, N., Margenat, P., & Visca, J. (2019, December). Robotito: programming robots from preschool to undergraduate school level. In 2019 19th International Conference on Advanced Robotics (ICAR) (pp. 296-301). IEEE.
- [6] Logo Foundation (2014) What is Logo.
https://el.media.mit.edu/logo-foundation/what_is_logo
- [7] Logo Foundation (2014) Logo History
https://el.media.mit.edu/logo-foundation/what_is_logo/history.html
- [8] Audrey W. (2015) Lego Mindstorms: A History of Educational Robots.
<http://hackeducation.com/2015/04/10/mindstorms>
- [9] Makeblock (2018) Mbot
<https://www.makeblock.com/steam-kits/mbot>
- [10] Modular Robotics (s.f.) Cubelets getting started
<https://www.modrobotics.com/cubelets/cubelets-getting-started/>
- [11] Luxrobo (s.f.) Modi
<https://modi.luxrobo.com>
- [12] García, J., & García, S. (2020). Las tecnologías en (y para) la educación. Montevideo, Uruguay: FLACSO Editorial. Recuperado el, 28.
- [13] García J., (2013) Hacia la masificación de la robótica educativa, Montevideo, Uruguay: Academia.edu. Recuperado de:
https://www.academia.edu/10899403/Hacia_la_masificación_de_la_robótica_educativa

- [14] INCO (fing) Butia Wiki
<https://www.fing.edu.uy/inco/proyectos/butia/>
- [15] RO-BOTICA, (2021) Robot Lilliput Naranja.
<https://ro-botica.com/Producto/Robot-Lilliput-Naranja/>
- [16] Pedersen, B. K. M. K., Andersen, K. E., Köslich, S., Sherzai, F., & Nielsen, J. (2018, March). Towards playful learning and computational thinking—Developing the educational robot BRICKO. In 2018 IEEE Integrated STEM Education Conference (ISEC) (pp. 37-44). IEEE.
- [17] Lego (s.f.) Mindstorms EV3 Specs,
<https://www.lego.com/en-us/product/lego-mindstorms-ev3-31313>
- [18] Kidstoy (s.f.) The LEGO MINDSTORMS NXT
<https://kidstoy.nop-station.com/the-lego-mindstorms-nxt-idea-book-design-invent-and-build-paperback-august-28-2007>
- [19] ROS (2021) Robot Operating System
<https://www.ros.org/>
- [20] AiBrain (2012) SIRO: multi-purpose smartphone robot
<http://aibrain.asia/?p=95>
- [21] SK Telecom (2017) SK Telecom’s Smart Robot Coding School Wins Global Corporate Award at ITU Telecom World Awards 2017
https://www.sktelecom.com/en/press/press_detail.do?idx=1234
- [22] MINT (s.f) The robobo project.
<http://education.theroboboproject.com/en/>
- [23] Simarobot (s.f) Simarobot
<https://web.simarobot.com/>
- [24] Kote Puerto (2012) Bandai Smart Pet
<https://www.xataka.com/robotica-e-ia/bandai-smartpet-robot-estilo-aibo-con-corazon-iphone>
- [25] Meikinit (2022) Robert Albert BT School,
<https://meikinit.com/producto/robot-albert-bt-school/>
- [26] Gerosa Barboza, A. (2021). Playing with robots to understand computational thinking in children’s cognitive development.
- [27] Siciliano, B., Khatib, O., & Kröger, T. (Eds.). (2008). Springer handbook of robotics (Vol. 200). Berlin: springer.
- [28] Alur, R., Kannan, S., & Yannakakis, M. (1999, July). Communicating hierarchical state machines. In ICALP (Vol. 99, pp. 169-178).

- [29] Estudio Comscore, El observador (2016) El uso de smartphones y tablets se disparo en uruguay,
<https://www.elobservador.com.uy/nota/el-uso-de-smartphones-y-tablets-se-disparo-en-uruguay-20164151080>
- [30] Comparative analysis of Android and iOS from security viewpoint. Computer Science Review, 40, 100372.
- [31] Norbert Kamienski (2020) React Native vs Ionic and Cordova: Comparison
<https://pagepro.co/blog/react-native-vs-ionic-and-cordova-comparison/#introduction>
- [32] Lionel Sujay Vailshery (2022) Cross-platform mobile frameworks used by software developers worldwide from 2019 to 202
<https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/>
- [33] Google (2022) Barcode scanning
<https://developers.google.com/ml-kit/vision/barcode-scanning>
- [34] IAndroid (2023) ImageFormat
<https://developer.android.com/reference/android/graphics/ImageFormat>
- [35] Google (2022) Image labeling
<https://developers.google.com/ml-kit/vision/image-labeling>
- [36] Google (2022) InputImage
<https://developers.google.com/android/reference/com/google/mlkit/vision/common/InputImage>
- [37] Google Cloud (2022) Vision AI
<https://cloud.google.com/vision?hl=es>
- [38] Gsmarena (2000) Samsung Galaxy Note 4
https://www.gsmarena.com/samsung_galaxy_note_4-6434.php
- [39] MIT (s.f.) About Scratch
<https://scratch.mit.edu/about>