UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

Diseño y desarrollo de una plataforma física, interactiva, y portátil para modelación urbana con fines educativos.

POR

Francisco Ignacio Valdés Rojas

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil en Telecomunicaciones.

Profesor Guía
Gabriel Saavedra Mondaca

Concepción,
17 de octubre de 2024

Agradecimientos

Mi vida universitaria ha sido un gran camino , lleno de experiencias, alegrías, logros, penas, fracasos, pero sobretodo de gran aprendizaje, estos siete años han pasado volando; De pasar a ser el joven de 18 años que entra a la universidad sin idea de nada (ni siquiera de la carrera que eligió), a la persona que soy hoy en días ha sido gracias a las personas que me han acompañado en todo mi proceso el cual no he vivido solo.

Primero agradecer a mis padres, los cuales, a pesar de nuestras diferencias, me han dado el apoyo desde que he nacido hasta el día de hoy.

A mis hermanos, quienes siempre han estado ahí, escuchándome y apoyando en momentos difíciles, y para poder sacarme una sonrisa.

Agradecer a los VDM, quienes conocí en pandemia, que a pesar de ser gente de distintas ciudades y de distintas edades, siempre me sacan una sonrisa al final del día cuando jugamos.

Agradecer también a todos los profesores de la universidad, sobretodo al profesor Gabriel Saavedra, por todo su apoyo y buena disposición en todo este tiempo.

Agradezco también a Citylab Bio Bio y a todo su equipo, que me han apoyado en la realización de este proyecto, y el llevar a cabo todas las ideas que surgieron en el camino.

Agradecer a Diego Ramirez, por guiarme en todo esto proceso y por su buena disposición frente a todas mis dudas.

Y finalmente agradecer a la tuna de la facultad de ingeniería de la universidad de concepción, que sin esta hermandad, nada de lo que he logrado hubiera sido posible .

Gracias por todo

Resumen

Citylab Biobío es un laboratorio de ciudad que utiliza una mesa con el sistema cityscope, el cual proyecta diversas métricas de estudio. Este sistema incluye placas interactivas que representan estados actuales y futuros; al cambiar estas placas, se modifican las métricas y, en consecuencia, la proyección. Dada la naturaleza del proyecto, el laboratorio participa de manera constante en diversas actividades, como exposiciones, talleres y presentaciones. Sin embargo, ha surgido la necesidad de llevar la mesa interactiva con el sistema cityscope a estos eventos, lo que ha resultado complicado debido al funcionamiento del sistema, que emplea cámaras, y al tamaño de la mesa.

En este contexto, se desarrolló una versión portátil del cityscope que reemplaza el sistema de detección de placas, pasando de cámaras a sensores RFID. Se construyó un nuevo backend utilizando una nueva base de datos desarrollada por el equipo de ciencia de datos del laboratorio, y se empleó Mapbox para la proyección de estos datos.

Los principales resultados incluyen el correcto funcionamiento del nuevo sistema, la proyección adecuada de los mapas y la interactividad lograda en la proyección al cambiar las placas entre los estados actual y futuro. Además, se facilitó el cambio de proyección entre diferentes categorías de estudio.

Abstract

Citylab Biobío is a city laboratory that uses a table with the Cityscope system, which projects various study metrics. This system includes interactive plates that represent current and future states; changing these plates alters the metrics and, consequently, the projection. Given the nature of the project, the laboratory constantly participates in various activities such as exhibitions, workshops, and presentations. However, there has been a need to bring the interactive table with the Cityscope system to these events, which has proven to be complicated due to the system's use of cameras and the size of the table.

In this context, a portable version of the Cityscope was developed, replacing the plate detection system, moving from cameras to RFID sensors. A new backend was built using a new database developed by the laboratory's data science team, and Mapbox was employed for the projection of these data.

The main results include the correct functioning of the new system, the proper projection of the maps, and the interactivity achieved in the projection when switching the plates between the current and future states. Additionally, the projection switch between different study categories was facilitated.

Índice General

Agradecimientos I

Resumen II

Abstract III

Índice de Figuras V

Índice de Tablas VIII

1. Introducción **1**

2. Definición del problema **2**

2.1. Funcionamiento Cityscope. 2

2.2. Definición del problema. 8

2.3. Objetivo general 9

2.4. Metodología 9

2.5. Alcances y limitaciones 11

3. Desarrollo **12**

3.1. Introducción 12

3.2. Soluciones propuestas por otros autores 12

3.3. Diseño,construcción física de mesa, y maqueta del sector costanera. 14

3.4. Elección de componentes y diseño circuito de sensores. 17

3.5. Desarrollo sistema de proyección y detección de mapas. 23

3.6. Programación del circuito de sensores RFID. 36

3.7. Automatización y despliegue de sistema en Raspberry pi. 37

3.8. Resultados 40

3.9. Tablas 44

4. Conclusión **46**

Índice de Figuras

2.1. Despliegue sistema Cityscope, junto con sus componentes. 2

2.2. Vista inferior mesa Cityscope. 3

2.3. Proyección de imagen mesa Cityscope. 3

2.4. Mesa interactiva con modelo sector costanera de Concepción. 4

2.5. Imágenes representativas de estudios sector costanera de Concepción. 4

2.6. Ubicación placas interactivas en el sector Costanera de Concepción. 5

2.7. Cambio proyección y gráficas en placas actual y futura. 6

2.8. Marcadores ArUco. 7

2.9. Funcionamiento general detección de placas Citylab Bío Bío. 7

2.10. Falla en la detección de marcadores ArUco. 8

2.11. Lector y Tags RFID. 9

3.1. Marcador ArUco cerrado. 13

3.2. Tablero de calibración para marcadores ArUco. 13

3.3. Detección marcadores ArUco haciendo uso de Yolo. 14

3.4. Viste superior mesa con medidas. 15

3.5. Viste frontal mesa con medidas. 15

3.6. Ubicación de las 12 placas en el sector costanera. 16

3.7. Modelo placa 1 en estado actual y futuro. 16

3.8. RFID Tag ubicado debajo de placa. 17

3.9. Modelo físico sector Costanera de Concepción 17

3.10. M5Stack básic core development kit. 18

3.11. M5Stack RFID sensor. 19

3.12. M5Stack PaHub2 Unit. 20

3.13. Raspberry pi 4 modelo B. 20

3.14. Diagrama esquemático circuitos RFID. 22

3.15. Posición sensores RFID en la mesa con sus respectivos canales. 23

3.16. Llaves base de datos Redis. 24

3.17. Explicación mapa a proyectar, construido en base al mapa actual y futuro. . . . 27

3.18. Diagrama de flujo backend. 29

3.19. Mapbox inciado y centrado en sector Costanera. 31

3.20. Http requests enviadas por Postman. 32

3.21. Mapa futuro proyectado en Mapbox. 33

3.22. Mapa actual proyectado en Mapbox. 34

3.23. Mapa combinado. 35

3.24. Diagrama final del sistema. 35

3.25. Menú encargado de seleccionar la categoría del mapa a proyectar. 37

3.26. Contenido archivo dockerfile. 38

3.27. Contenido archivo docker-compose. 39

3.28. Proyector desplegando Mapbox sobre la mesa. 40

3.29. Mapa de tramites en estado actual. 41

3.30. Cambio de placa 1 Actual a placa 1 Futuro. 42

3.31. Cambio de placa 4 Actual a placa 4 Futuro. 43

3.32. Mapa de tramites con placa 1 y placa 4 en escenario futuro. 44

Índice de Tablas

3.1. Tabla de componentes electrónicos 44

3.2. Tabla de precios componentes. 44

3.3. Tabla comparativa sistemas 45

Acrónimos

API Application Programming Interface.

HTTP Hypertext Transfer Protocol.

IoT Internet of Things.

JSON JavaScript Object Notation.

KHz Kilohertz.

MHz Megahertz.

RFID Radio-frequency identification.

UID Unique Identifier.

1. Introducción

Concepción es una comuna, perteneciente al área metropolitana del Gran Concepción, en donde el núcleo urbano ejerce un significativo impacto en el comercio nacional. Se caracteriza también por tener varios puentes, parques, lagunas y universidades [1], lo cual hace que se concentre varios tipos de servicios dentro de esta ciudad, como educacionales, cultura, habitacional, etc. Bajo este contexto, se inauguró Citylab Bío Bío, el cual es un laboratorio de ciudad, que pertenece a la red Mit city science lab, que se encuentra en el Mit media lab, con distintos laboratorios en ciudades como Hamburgo, Shangai,Taipei, Ciudad de Ho Chi Minh, Toronto, Andorra y Guadalajara, en donde su objetivo es contribuir a la toma de decisiones en la planificación urbana, mediante la implementación de herramientas y metodologías que permiten realizar simulaciones a escala, creada a partir de datos obtenidos por comunidades y actores relevantes[2]. Estos datos son modelados de forma interactiva a través de la mesa "Cityscope", que permite la simulación a escala de situaciones concretas, y hacer el análisis de cómo afectan en escenarios actuales y futuros, donde su principal zona geográfica de estudio es el sector Costanera de Concepción (Pedro del Río Zañartu, Aurora de Chile, Pedro de Valdivia)[3]. Esta plataforma desarrollada por Citylab Bío Bío, para lograr la interactividad de las placas que representan zonas geográficas de interés en el sistema, hace uso de cámaras, que apuntan desde de la parte de abajo a la mesa, y dependiendo los cambios que detecte en sus placas, envía la información al backend del sistema. Este informe se centrará en entender el funcionamiento del sistema Cityscope usado en la mesa interactiva, los problemas que ha tenido el uso de cámaras en el sistema, sus posibles soluciones, asi como la creación de la mesa Cityscope en su versión portatil, haciendo uso de sensores RFID, junto con todo su desarrollo correspondiente.

2. Definición del problema

2.1. Funcionamiento Cityscope.

Actualmente, para poder hacer el despliegue de la mesa interactiva, se puede hacer el uso del sistema "Cityscope", desarrollado por MIT Media Lab City Science group, el cual es de código abierto, que nos permite visualizar en la mesa, de forma interactiva, los distintos indicadores y gráficas, que permiten comprender el impacto de diversos cambios que hay dentro de las ciudades.[4]

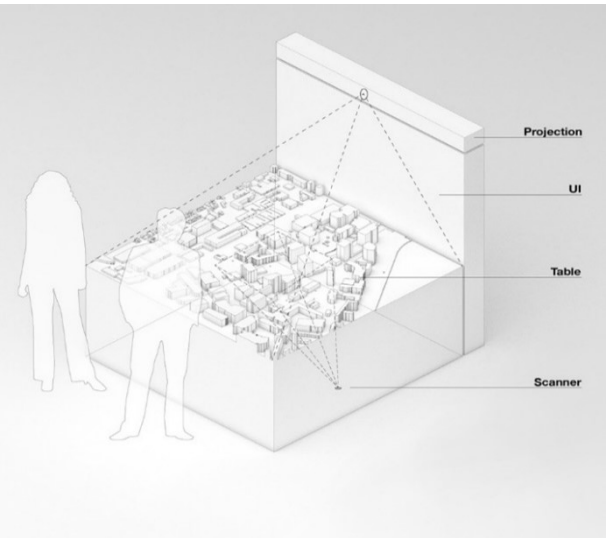


Fig. 2.1: Despliegue sistema Cityscope, junto con sus componentes.

Como se muestra en la Figura 2.1, el despliegue se hace usando una mesa, la cual debe tener encima de ella un modelado físico de la ciudad, o zona, que se quieren hacer los distintos estudios, en donde este modelo físico tendrá placas intercambiables de ciertos lugares donde se quieren mostrar los cambios e impactos de las construcciones modeladas en cada placa; También se hace uso de un proyector, el cual va a ser el encargado de emitir imágenes de la ciudad en la mesa, con rectángulos de colores, que indican alguna métrica de estudio que afecta en la zona, las cuales irán cambiando en la zonas de las placas al hacer un intercambio de estas; Y por último una pantalla, que ira mostrando gráficas, sobre las diferentes métricas de estudios, las cuales irán cambiando al unísono de las placas.

Para poder hacer la detección, se hace el uso de Legos en la mesa, como se muestra en la Figura 2.2, ubicados en donde se quieren hacer los cambios en la proyección de la imagen de la mesa.

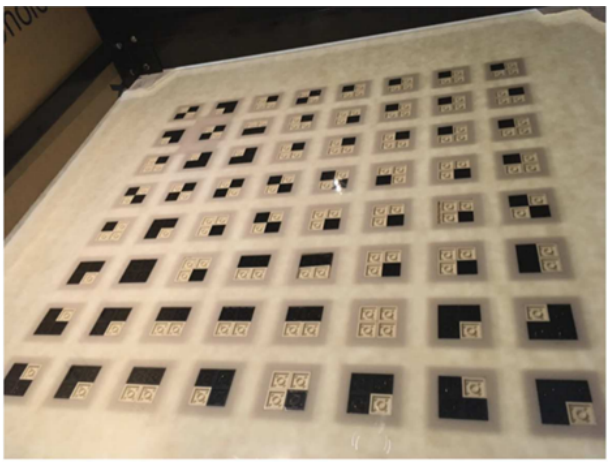


Fig. 2.2: Vista inferior mesa Cityscope.

Debajo de la mesa va puesta una cámara que va escaneando las diferentes combinaciones de forma y/o colores que tienen los cuadrados de Lego, en donde dependiendo de los bloques blanco y negro que se detecten en cada cuadrado de Lego, va a enviar la información de ese caso al backend del sistema, y se proyectará la imagen correspondiente a ese caso[5], como se ve en la Figura 2.3.

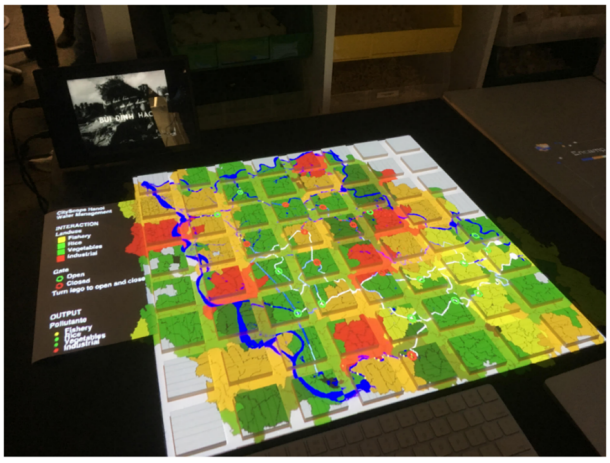


Fig. 2.3: Proyección de imagen mesa Cityscope.

En Citylab Bío Bío, para poder implementar el sistema Cityscope, se creó una mesa interactiva siguiendo los mismos objetivos y funcionalidades, con variaciones en la maqueta y métricas de estudio, adaptándolas en el contexto de Concepción y el sector de estudio.

Fig. 2.4: Mesa interactiva con modelo sector costanera de Concepción.

La Figura 2.4, muestra la mesa interactiva, en donde se encuentra modelado a través de piezas Legos el sector costanera de Concepción, en donde Citylab Bío Bío realiza distintos estudios de diferentes proyectos públicos, y ven su impacto en diferentes ámbitos, como áreas verdes, cultura, espacios públicos, deporte, residencial, diversidad de suelos, densidad de población; Los cuales se representan a través de imágenes como muestra la Figura 2.5, en donde la escala de colores representa la intensidad del impacto de las diferentes construcciones en la zona.

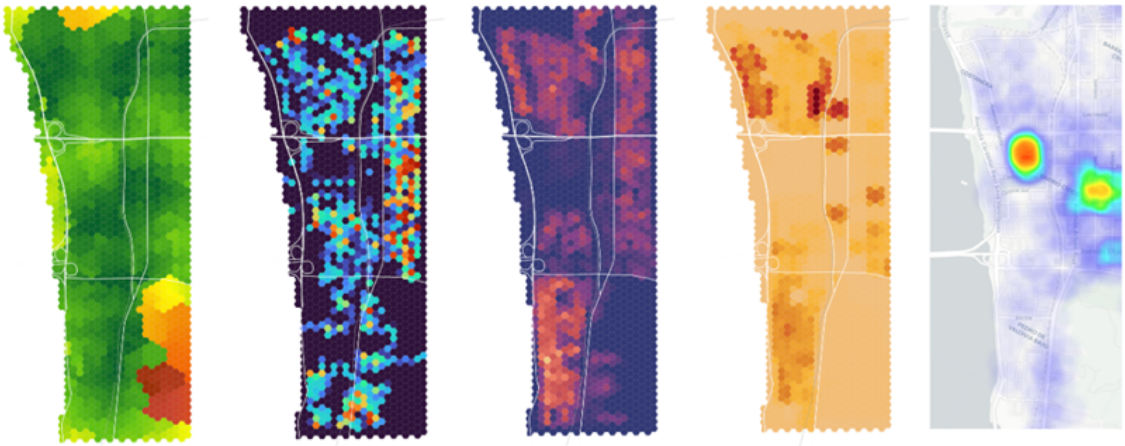
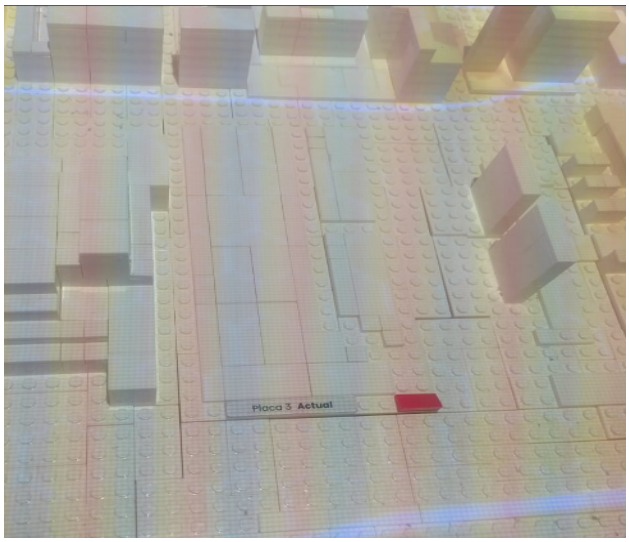


Fig. 2.5: Imágenes representativas de estudios sector costanera de Concepción.

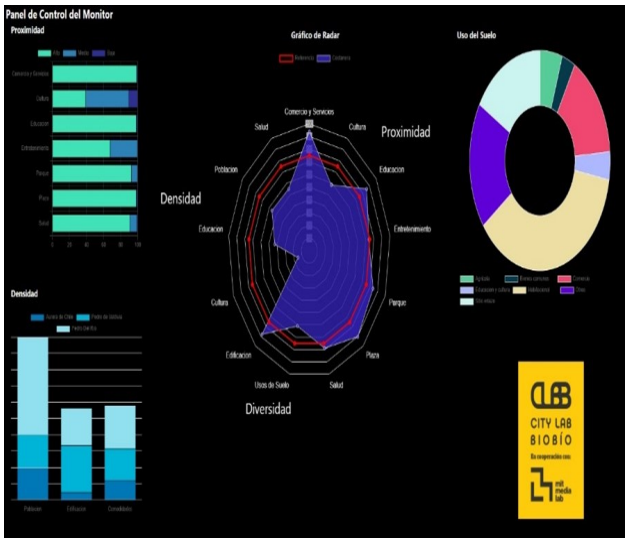
En la mesa interactiva están ubicadas 7 placas, las cuales van posicionadas en 7 zonas del sector, como se muestra en la Figura 2.6, en donde cada zona va puesta una placa que modela el escenario actual, o futuro, lo que dependiendo de cual este puesta, va variando la imagen proyectada a la mesa, representando en la escala de colores, el impacto que tiene la construcción, a la vez se muestra en la pantalla una gráfica con las diferentes medidas y métricas que afectan

estos cambios, como se muestra en la Figura 2.7, por lo que en total, se tienen 7 placas actuales y 7 placas futuro.

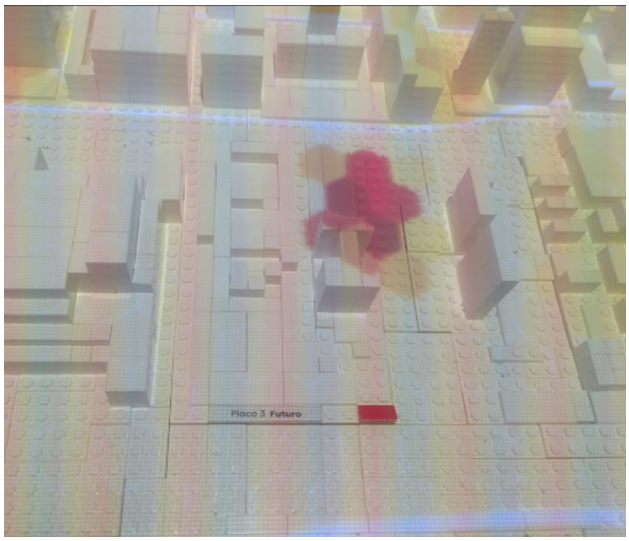
Fig. 2.6: Ubicación placas interactivas en el sector Costanera de Concepción.



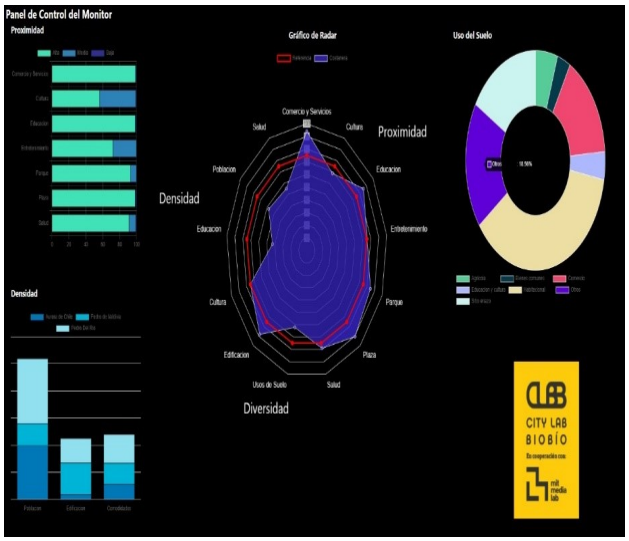
(a) Proyección imagen en zona de placa 3 actual.



(b) Gráficas para el mapa con placa 3 actual.



(c) Proyección imagen en zona de placa 3.



(d) Gráficas para mapa con placa 3 futuro.

Fig. 2.7: Cambio proyección y gráficas en placas actual y futura.

La forma en detección de las placas es usando cámaras y marcadores ArUco, para hacer los cambios en la proyección de la imagen, a la hora de detectar cambios en la placa. ArUco es una librería basada en OpenCV, lo cual permite generar marcadores, como se ven en la Figura 2.8, los cuales son cuadrados negro que dentro tienen una matriz binaria[6], que se pueden asignar un identificador para poder diferenciarlos con una cámara, estos cuadrados pueden ser generados variando su tamaño e identificador, dependiendo de la aplicación que se le dé.

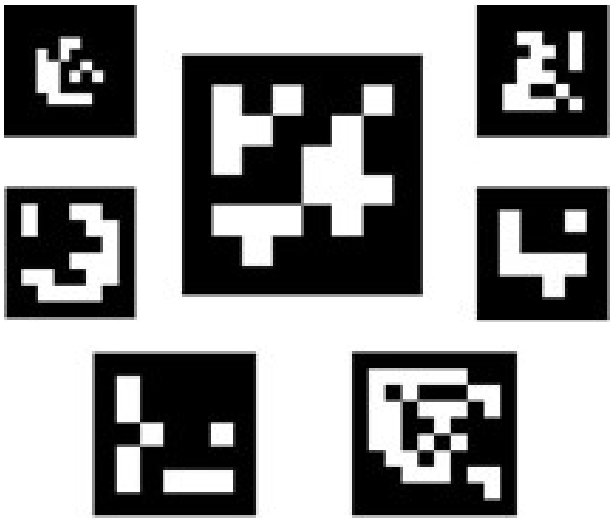


Fig. 2.8: Marcadores ArUco.

Como se muestra en la Figura 2.9. Estos marcadores ArUco van pegado debajo de las placas, que son detectados por dos cámaras, que están conectadas a un raspberry pi, estos componentes se ubican debajo de la mesa y apuntan a esta, luego, la raspberry pi ejecuta un programa que detecta los marcadores, los cuales están asociados a números, los guarda en un arreglo de 7 valores. Después, este arreglo lo envía al backend del sistema mediante el método HTTP Get, y dependiendo del arreglo que recibe el backend, proyecta la imagen correspondiente a la mesa, logrando así la interactividad de las placas.

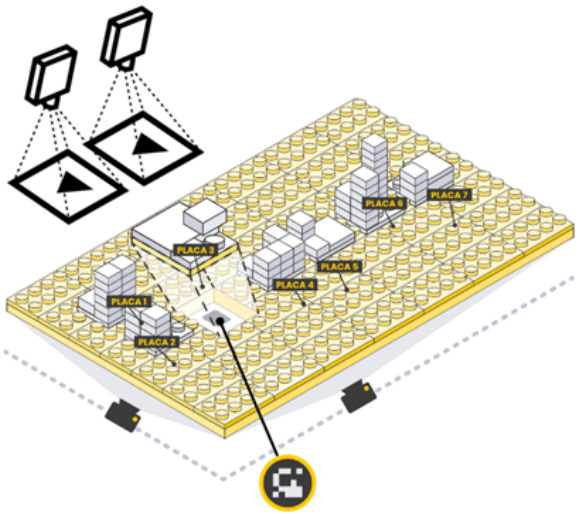


Fig. 2.9: Funcionamiento general detección de placas Citylab Bío Bío.

2.2. Definición del problema.

El método de detección basado en imágenes y cámaras, descrito anteriormente, puede enfrentar varios problemas al reconocer los marcadores debido a condiciones externas que afectan a las cámaras. Factores como la iluminación del entorno, la descalibración de las cámaras o cambios en su posición por movimientos inesperados, pueden comprometer la detección precisa de los marcadores, como se muestra en la Figura 2.10. Estas dificultades impiden la transmisión correcta de la información de las placas, lo que resulta en fallos del sistema al no poder proyectar la imagen correspondiente en la mesa. Esto complica el uso del sistema en lugares fuera del laboratorio.

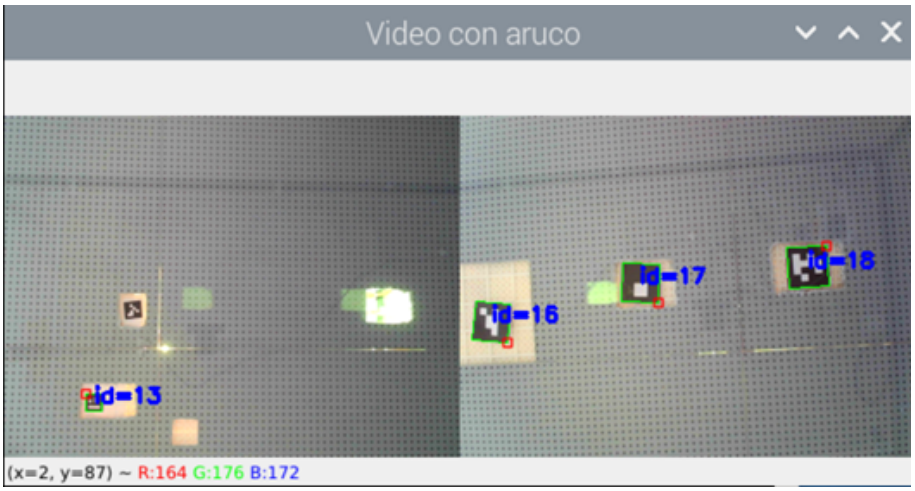


Fig. 2.10: Falla en la detección de marcadores ArUco.

Desde la creación de Citylab Bio Bio, ha surgido la necesidad de poder llevar este sistema a diferentes lugares, como exposiciones, colegios, empresas, etc., pero debido al funcionamiento del sistema, y los problemas de detección de este, no ha sido posible, por lo que este proyecto buscó crear la mesa cityscope en su versión portátil, cambiando su forma de detección, backend, y despliegue del sistema.

Para mejorar la comunicación entre las placas y el backend del sistema, se propuso el uso de la tecnología RFID. RFID es una tecnología de comunicación inalámbrica que permite a un lector capturar datos codificados en tags RFID mediante ondas de radio. Esta tecnología se compone de un lector y un tag, tal como se ilustra en la Figura 2.11. Los rangos de lectura dependen de la frecuencia del lector, pudiendo variar entre 860 MHz y 960 MHz (hasta 12 metros), 13.56 MHz (10 cm hasta 1 metro), y entre 125 kHz y 134 kHz (hasta 10 cm) [7].

(a) Lector RFID.

(b) RFID Tags.

Fig. 2.11: Lector y Tags RFID.

Citylab Bío Bío ha recopilado nuevas bases de datos y mapas actualizados, lo que ha motivado el desarrollo de un nuevo backend diseñado para procesar esta información. Este backend se implemento en una Raspberry Pi, que se encargará de gestionar y procesar los datos en tiempo real. Además, la Raspberry Pi esta conectada a un proyector que apuntará a la mesa, siendo responsable de proyectar los mapas de manera dinámica sobre ella.

Todos estos componentes van a estar conectados a un Access Point generado por un celular, para facilitar la comunicación entre ellos.

2.3. Objetivo general

Diseño y desarrollo de una plataforma física, interactiva y portátil, basado en el sistema Cityscope, para el despliegue y proyección de diferentes modelamientos y estudios del sector costanera en la mesa interactiva, mediante el uso placas de desarrollo y sensores RFID, que detecten los cambios hechos en la maqueta, y se comuniquen con el backend del sistema, que sera previamente desarrollado, implementado en un raspberry pi, logrando la interactividad, y priorizando la portabilidad, para poder hacer el sistema Cityscope transportable , con el fin de hacer presentaciones en muestras, exposiciones, talleres , juntas de vecinos,etc...

2.4. Metodología

- **Diseño,construcción física de mesa, y maqueta del sector costanera.**

Para poder hacer el diseño, se uso un software de modelado 3D para diseñar la mesa, con todas las medidas y materiales especificados, teniendo en consideración su portabilidad. La construcción de esta se hizo con palos de madera y una plancha de acrílico en donde ira la maqueta encima de ella; Luego, a través de piezas Legos, y basándonos en el modelo construido en laboratorio, se hizo el modelo del Cityscope portátil del sector costanera de Concepción.

- **Elección de componentes y diseño circuito de sensores.**

  Para lograr este objetivo, se investigó cual será la mejor opción del componente para su posterior compra de todos los componentes electrónicos necesarios, para poder hacer el despliegue de la mesa y el sistema Cityscope, detallando las unidades, y precios de estos en el momento de la compra, para finalmente hacer el diseño esquemático del circuito de sensores RFID, especificando sus conexiones y la forma que tendrá este.

- **Desarrollo sistema de proyección y detección de mapas.**

  Para hacer el despliegue del sistema, se hizo uso de los recursos adquiridos recientemente por el laboratorio, como Mapbox y la nueva base de datos, para desarrollar un nuevo sistema de detección de mapas. Este sistema estará dividido en dos partes: un Frontend y un Backend, que se comunicarán a través de APIs.

- **Programación del circuito de sensores RFID.**

  Para lograr este objetivo, se usó los sensores RFID, la placa de desarrollo cotizada anteriormente, y un compilador de software para poder programar el circuito, y crear todas las funcionalidades necesarias para la comunicación del backend con el circuito, así como un menú para poder seleccionar el mapa a proyectar.

- **Automatización y despliegue de sistema en Raspberry pi.**

  Una vez completados los objetivos anteriores, se automatizó el sistema mediante el uso de contenedores Docker. Estos contenedores también incluyen el despliegue de la base de datos, permitiendo que el sistema pueda ser implementado en diferentes entornos según sea necesario. En este caso particular, el despliegue se realizó en una Raspberry Pi, donde se ejecutarán los contenedores creados.

- **Resultados.**

Finalmente ya con todos los puntos anteriores completados, se hicieron múltiples pruebas con la mesa ya creada,junto con sus resultados, con el fin de comprobar el funcionamiento de todos sus elementos, para que este sistema pueda ser desplegado en futuras presentaciones.

2.5. Alcances y limitaciones

Al desarrollar una plataforma portátil que integra componentes electrónicos, es crucial tener en cuenta la disponibilidad de energía en los lugares donde se instalará la mesa. Idealmente, será necesario contar con una toma de corriente para alimentar los componentes electrónicos. Como alternativa, se puede considerar el uso de baterías para energizar el sistema, pero es importante señalar que tanto el raspberry pi como el proyector deberán estar conectados a una fuente eléctrica. Por lo tanto, el funcionamiento de la mesa dependerá de la disponibilidad de electricidad en el lugar de despliegue.

Asimismo, se recomienda que durante el despliegue en diferentes ubicaciones esté presente una persona capacitada en el funcionamiento del sistema. Esta persona será responsable de ejecutar los procedimientos necesarios para la correcta instalación del Cityscope y resolver cualquier problema que pueda surgir durante su uso.

Por último, es esencial que el lugar donde se instale el sistema disponga de acceso a internet, ya que se utilizará un Access point creado a través de un teléfono móvil para facilitar la comunicación entre los distintos componentes del sistema.

3. Desarrollo

3.1. Introducción

Para desarrollar la mesa interactiva con el sistema Cityscope, es fundamental comprender el objetivo de implementar tecnología RFID en la detección de placas. La idea es reemplazar el sistema actual de cámaras con marcadores ArUco por un sistema basado en sensores RFID, tags RFID, y una placa de desarrollo que ejecutará el software encargado de leer y transmitir la información de los tags RFID.

Este nuevo enfoque propone que cada placa de la mesa esté equipada con un RFID Tag, mientras que los sensores se ubiquen en las posiciones correspondientes a cada placa. De esta manera, cuando se cambie una placa por otra, el sensor detectará el RFID Tag y determinará si corresponde a una placa actual o futura, enviando ese estado al backend del sistema. El backend, a su vez, se encargará de proyectar en la mesa el mapa correspondiente a la categoría seleccionada para el análisis.

Todos los componentes del sistema estan conectados a un Access Point generado por un teléfono móvil, el cual se utiliza para enrutar la información y permitir la comunicación entre los dispositivos.

3.2. Soluciones propuestas por otros autores

Frente a los problemas mencionados, otros autores han explorado varias formas de poder mejorar la precisión en la detección de marcadores ArUco con cámaras, como el uso de marcadores cerrados como se ve en la Figura 3.1, que permite una localización de las esquinas de los marcadores más precisa, y así mejorar la detección de estos[8]. Como también el uso de un tablero de calibración, en el cual se imprimen varios de estos marcadores,como se ve en la Figura 3.2, en donde se tomarán imágenes desde varias perspectivas del tablero y se ejecutan funciones disponibles dentro de la librería, que reciban estas imágenes y se logra la calibración[9].

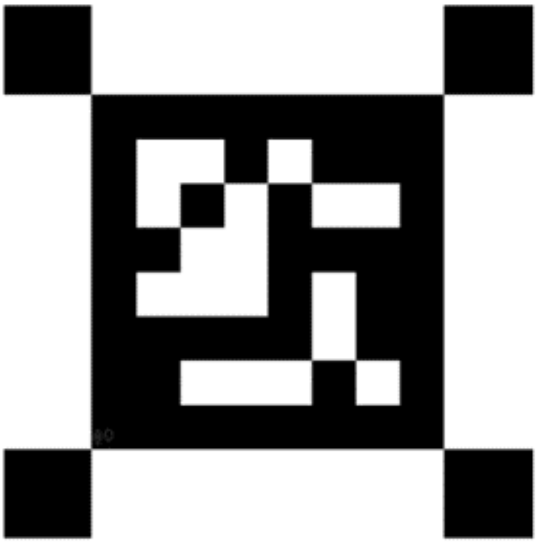


Fig. 3.1: Marcador ArUco cerrado.

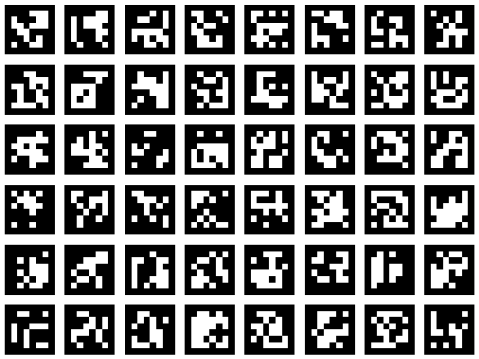


Fig. 3.2: Tablero de calibración para marcadores ArUco.

Otra solución en la mejora de la detección de marcadores es hacer el uso de filtros, como el filtro Kalman, el cual es un algoritmo que permite estimar el estado en tiempo discreto de un proceso, a través de sistemas lineales y ruidos, puede proporcionar estimaciones; Este filtro se aplicaría antes para poder hacer una predicción del marcador, y después de las mediciones, el filtro se va actualizando de los marcadores ArUco.[10]

Por último, también está la posibilidad de usar redes neuronales convolucionales para mejorar la detección de marcadores ArUco, en la cual, a través de distintas redes neuronales, como you only look once network (YOLO), y Spatial Pyramid Pooling(SPP) network se va entrenando un modelo mostrando distintos marcadores, variando su distancia y perspectiva, alcanzando una

tasa de detección superior al 90 % en distancias sobre 7 metros, como se ve en la Figura 3.3.[11]

Fig. 3.3: Detección marcadores ArUco haciendo uso de Yolo.

3.3. Diseño,construcción física de mesa, y maqueta del sector costanera.

Para iniciar la construcción de la mesa, se considero el espacio disponible para su ubicación. El objetivo principal es crear una plataforma sobre la cual se pueda construir una maqueta del sector costanera de Concepción utilizando piezas Lego, por lo que se utilizó dos planchas cuadradas de Lego, cada una con un lado de 38.5 centímetros, dispuestas una al lado de la otra.

Teniendo en cuenta estas dimensiones, se procedio a diseñar la mesa utilizando el software SketchUp para visualizar claramente sus medidas. El diseño realizado se puede observar en las Figuras 3.4 y 3.5.

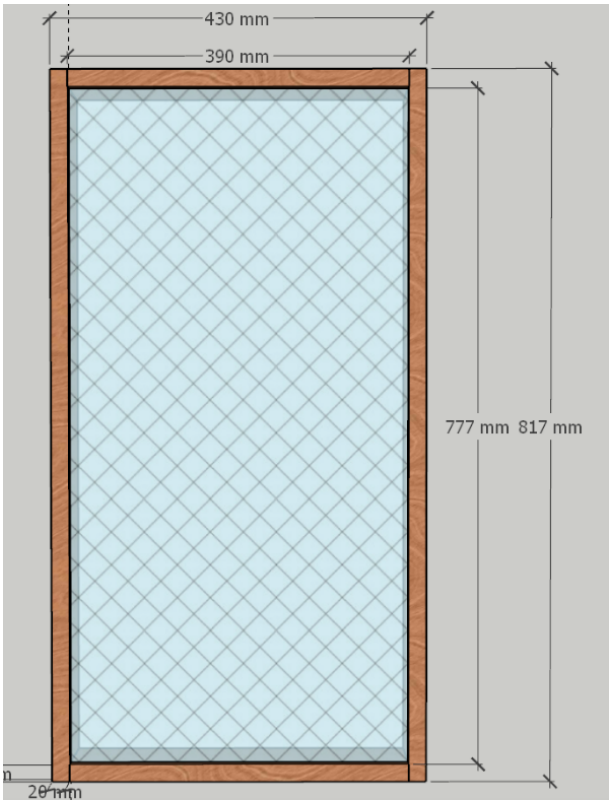


Fig. 3.4: Viste superior mesa con medidas.

Fig. 3.5: Viste frontal mesa con medidas.

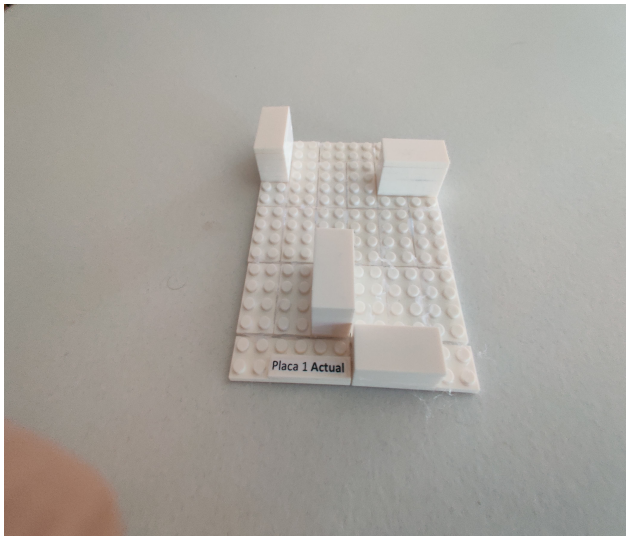
Estas presentan las dimensiones necesarias para la construcción de la mesa, la cual se realizó utilizando madera para la estructura y una lámina de acrílico transparente. Estos materiales se recomiendan por su facilidad de adquisición y manejo durante el montaje.

Una vez construida la mesa, se procedio a crear el modelo físico del sector Costanera de Concepción. Cabe señalar que este modelo es una representación aproximada del área, y no una réplica exacta de las calles y edificaciones actuales.

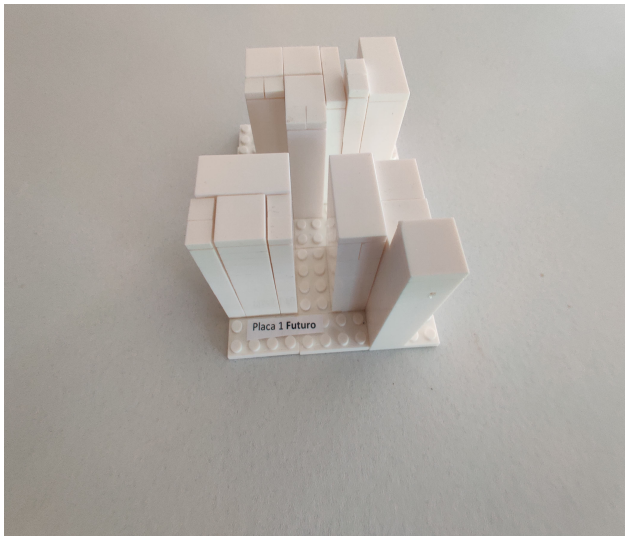
Para su elaboración, se delimitaron las zonas donde se ubicarán las placas, considerando los estudios en curso en el laboratorio y los resultados proyectados a futuro. Se optó por utilizar un total de 12 placas, cuya distribución se muestra en la Figura 3.6.

Fig. 3.6: Ubicación de las 12 placas en el sector costanera.

Ya con las ubicación de las placas definidas, se procedio a hacer la construcción de ellas en donde, en cada una de esas placas, se tiene que construir una simplificación basada en legos de un estado presente,y un estado futuro, como se ve en la Figura 3.7 donde el estado presente serian las construcciones actuales, y el estado futuro serian nuevas construcciones que irían en el sector donde esta la placa, lo que afectaría la accesibilidad de diferentes servicios en el sector.



(a) Modelo correspondiente a la placa 1 en un estado actual.



(b) Modelo correspondiente a la placa 1 en un estado futuro.

Fig. 3.7: Modelo placa 1 en estado actual y futuro.

Con los modelos de las placas ya construidos, se procedio a colocar un RFID tag debajo de

cada una, tal como se muestra en la Figura 3.8, con el propósito de diferenciarlas entre sí. El método de detección de estos tags será detallado más adelante.

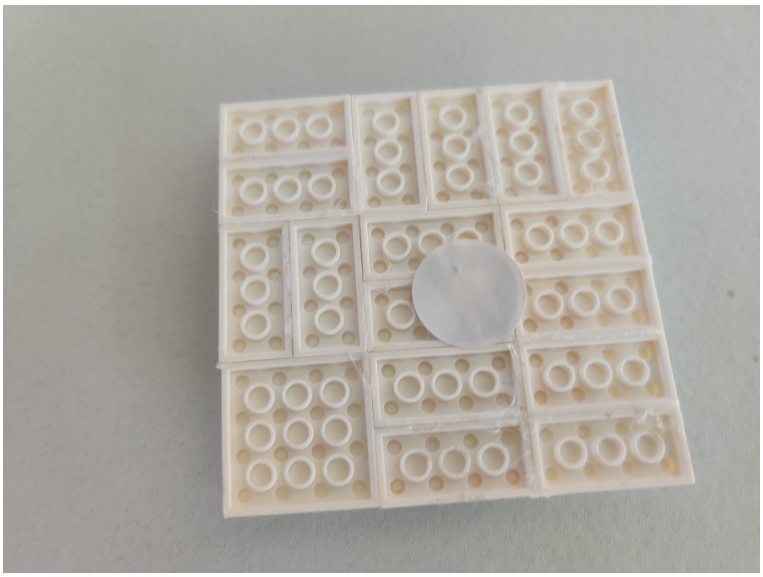


Fig. 3.8: RFID Tag ubicado debajo de placa.

Finalmente, ya con todas las placas construidas, se terminó el modelo del sector Costanera de Concepción el cual se ve en la Figura 3.9.

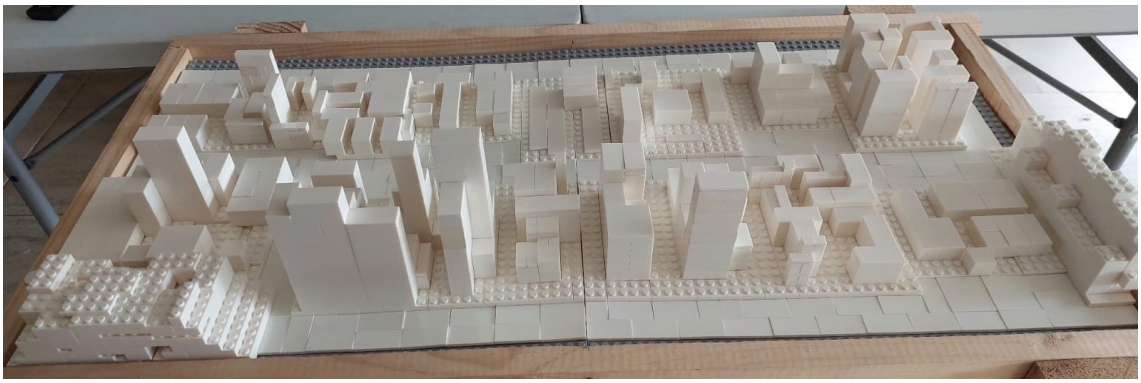


Fig. 3.9: Modelo físico sector Costanera de Concepción .

3.4. Elección de componentes y diseño circuito de sensores.

En el desarrollo del proyecto Cityscope portátil, es esencial evaluar los tipos de componentes a utilizar y sus costos. Para ello, se elaboró una lista detallada de todos los componentes necesarios,

la cual se presenta en la tabla 3.1.

Los componentes electrónicos enumerados son cruciales para la implementación del proyecto. Con esta lista disponible, se procedió a realizar la cotización, priorizando la relación calidad-precio para garantizar el correcto funcionamiento de la mesa interactiva.

En particular, para la selección de placas de desarrollo, sensores y multiplexores, se optó por productos de la marca M5Stack, un proveedor reconocido de soluciones IoT que desarrolla hardware tanto para usuarios particulares como para la industria [12].

La placa de desarrollo elegida para el proyecto es el ESP32 Basic Core IoT Development Kit V2.7, que se muestra en la Figura 3.10. Este dispositivo, basado en el desarrollo ESP32, cuenta con dos microprocesadores LX6 de 32 bits con una frecuencia de hasta 240 MHz. Incluye una pantalla IPS de 2 pulgadas, un altavoz y tres botones para la interacción, así como múltiples interfaces y pines programables [13].

En el momento de la compra, esta placa tuvo un costo de $39,90 dólares, lo que equivale a aproximadamente $37.787 pesos chilenos.

Fig. 3.10: M5Stack básic core development kit.

Para los sensores RFID, se cotizó la unidad RFID2, que se muestra en la Figura 3.11. Este sensor cuenta con un chip WS1850s, que utiliza comunicación I2C y opera a una frecuencia de 13.56 MHz, permitiendo una distancia de lectura de menos de 20 milímetros, lo cual es ideal para la detección de los RFID tags [14]. En el momento de la compra, el costo del sensor fue de

$4,95 dólares, lo que equivale a aproximadamente $4.687 pesos chilenos.

Fig. 3.11: M5Stack RFID sensor.

Considerando que se utilizó varios sensores RFID al mismo tiempo y que será necesario obtener las lecturas de la información que estos sensores proporcionen, se requerirá el uso de un multiplexor. Por lo tanto, se eligió el multiplexor PaHub2 Unit, que se muestra en la Figura 3.12. Esta unidad de expansión I2C permite utilizar 6 canales, en los cuales se pueden conectar múltiples sensores que utilizan comunicación I2C [15]. En el momento de la compra del componente, su costo fue de $7,95 dólares, lo que equivale a aproximadamente $7.529 pesos chilenos.

Fig. 3.12: M5Stack PaHub2 Unit.

Para la adquisición de la Raspberry Pi, se optó por el modelo Raspberry Pi 4 Modelo B con 8 GB de RAM, como se muestra en la Figura 3.13. Este dispositivo será el encargado de ejecutar todos los programas necesarios para el funcionamiento del sistema Cityscope. En el momento de la compra del componente, su costo fue de $139,990 pesos chilenos.

Fig. 3.13: Raspberry pi 4 modelo B.

Con respecto al proyector, se utilizó uno que ya está disponible en el laboratorio, por lo que no será incluido en el presupuesto.

Con la mayoría de los componentes ya adquiridos, se presenta el costo en pesos chilenos de cada uno, como se muestra en la tabla 3.2. Una vez que se han cotizado y comprado los componentes electrónicos, se procedio al diseño esquemático del circuito RFID. Para este diseño, es importante considerar el tipo de comunicación de los sensores, que utilizan el protocolo **I2C**. Este protocolo, basado en una arquitectura maestro-esclavo, permite la conexión de varios dispositivos con diferentes direcciones a través del mismo bus. Los pines que utiliza este protocolo son:

- **SDA(serial data line):** Es la línea de datos. Se utiliza para transmitir y recibir datos entre los dispositivos I2C. Todos los dispositivos en el bus I2C comparten esta línea para enviar y recibir bits de información de manera serial.
- **SCL (Serial Clock Line):**Es la línea de reloj. Esta línea es utilizada por el maestro (generalmente un microcontrolador o procesador) para sincronizar la transmisión de datos en el bus. El maestro genera los pulsos de reloj para coordinar el envío y la recepción de bits a través de la línea SDA.

En este caso, dado que se necesitan conectar 12 sensores RFID del mismo tipo, todos con la misma dirección, es necesario utilizar un multiplexor I2C para recibir y diferenciar la información de cada sensor, esto permite programarlos y configurarlos para su aplicación. El multiplexor se conectará a la placa de desarrollo M5Stack a través de su puerto A.I2C, que utiliza una conexión Grove. Este puerto facilita la conexión de dispositivos I2C a la placa mediante los pines **SCL**, **SDA**, **5V** y **GND**, los cuales son compartidos tanto por el multiplexor como por los sensores. El diagrama esquemático de esta configuración se muestra en la Figura 3.14.

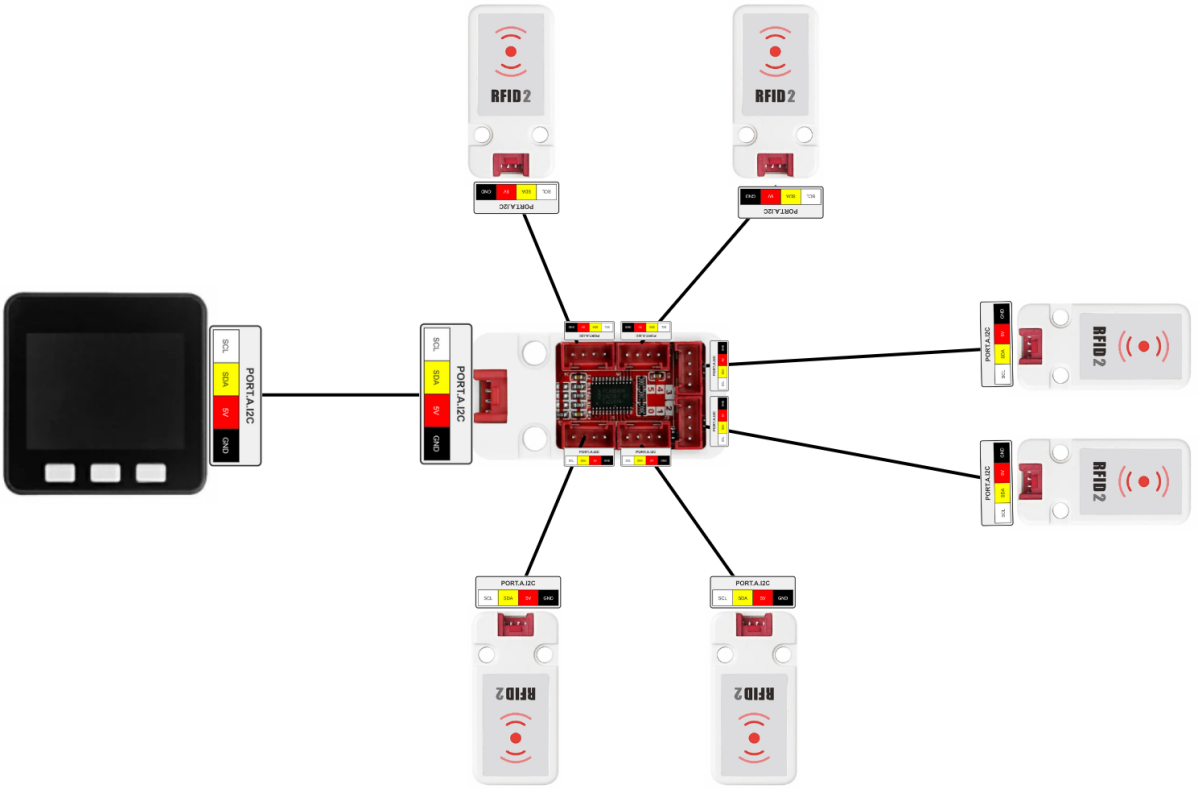


Fig. 3.14: Diagrama esquemático circuitos RFID.

Este circuito se repite por una segunda vez para poder conseguir 12 sensores, los cuales van ubicados en la mesa previamente construida y posicionados como se muestra en la Figura. 3.15

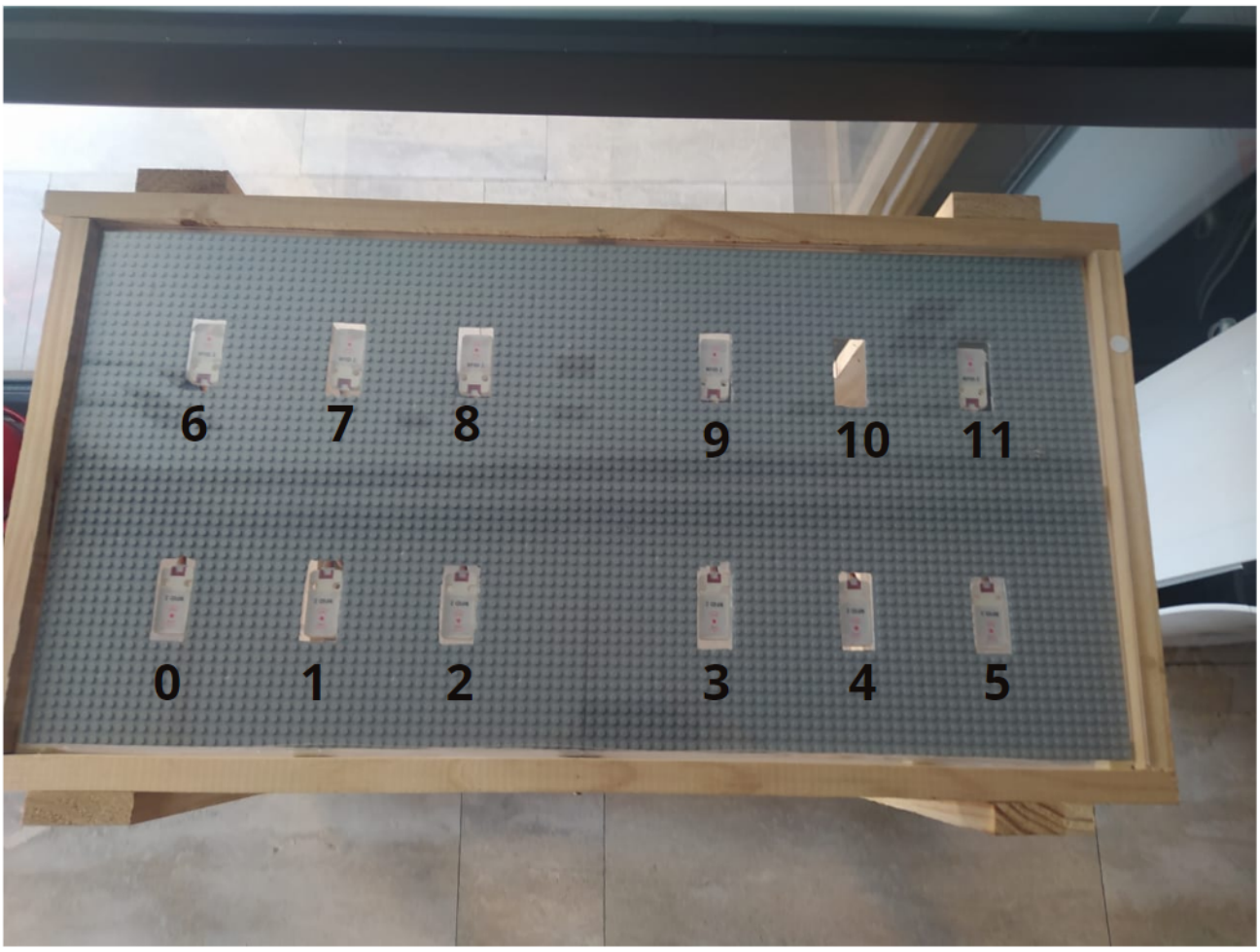


Fig. 3.15: Posición sensores RFID en la mesa con sus respectivos canales.

Los canales de lectura de cada circuito están enumerados de la siguiente manera: la primera placa leerá los canales del 0 al 5, mientras que la segunda placa se encargará de los canales del 6 al 11. Es importante destacar que ambos circuitos estarán energizados mediante una conexión USB a la Raspberry Pi.

3.5. Desarrollo sistema de proyección y detección de mapas.

Recientemente, el laboratorio ha concretado varios acuerdos con diversas organizaciones y municipalidades [16], lo que ha generado cambios en los datos, su recolección y las herramientas utilizadas para su visualización. Este nuevo sistema, que se implemento en el Cityscope portátil,

utilizó los datos recientes recopilados por el equipo de ciencia de datos del laboratorio. Como resultado, se modifico las formas de proyección e integrarán las herramientas adquiridas, que ahora están disponibles para el laboratorio.

Esta nueva base de datos, esta construida en base a la recolección de datos, calculando el tiempo de viaje de una determinada área hacia el punto de servicio mas cercano de una categoría de estudio, por ejemplo, tramites, salud deportes, etc.., estos datos hicieron a través de Open Street Maps[17], para red vial de peatones, y los puntos de interés de elaboración propia del equipo de ciencia de datos, ya con estos datos, a través de análisis de grafos, se calcula la distancia mas cercana que tiene hacia un servicio en especifico, en un área determinada.

En base a los estudios realizados previamente, el equipo de ciencia de datos construyó una base de datos **Redis**, diseñada para almacenar datos en memoria RAM, lo que permite un mayor rendimiento. Esta base de datos utiliza un modelo de datos basado en clave-valor [18]. En nuestro caso, se emplearon llaves para especificar las diferentes categorías en dos estados: actual y futuro, como se muestra en la Figura 3.16, donde el número junto a cada categoría define su estado, con 0 representando el estado futuro y 1 el estado presente.

Fig. 3.16: Llaves base de datos Redis.

Dentro de cada una de estas categorías, se almacenan datos en formato **GeoJSON**, el cual es un formato basado en **JSON** utilizado para representar datos geoespaciales, el cual es una notación de objetos de Javascript. Este formato permite describir objetos geométricos

como puntos, líneas y polígonos, junto con sus propiedades asociadas. En nuestro caso, los archivos GeoJSON contienen polígonos que construyen un mapa que abarca el sector costanera de Concepción. Cada polígono tiene una propiedad llamada **path_length**, la cual representa el resultado del estudio realizado por el equipo de ciencia de datos, mencionado anteriormente. Nos interesa utilizar este valor para crear una escala de colores, lo que permitirá una mejor representación visual en el mapa.

Con los datos y sus formas ya identificados, es necesario construir un sistema que procese estos datos GeoJSON, seleccione la categoría a proyectar y combine los valores de path_length en los puntos donde están ubicadas las placas, entre el mapa futuro y el presente de una misma categoría.

Para ello, a traves de diferentes convenios, el laboratorio logro tener acceso a una API key, que nos permite hacer uso de una librería de javascript llamada Mapbox GL JS la cual es una biblioteca de código abierto que permite la visualización y manipulación interactiva de mapas vectoriales en la web[19]. A diferencia de los mapas rasterizados tradicionales, que consisten en imágenes estáticas, los mapas vectoriales renderizados con Mapbox GL JS son altamente dinámicos y permiten una mayor flexibilidad en cuanto a la personalizaron y el estilo de los elementos que componen el mapa.

Para una mayor estructuración en este proyecto, se separa en dos partes, el backend y el frontend:

- **backend** es la parte del sistema que corresponde a el procesamientos de los datos,la infraestructura, y parte lógica del sistema; En este proyecto para el desarrollo del backend se uso el lenguaje de programación Python para el procesamiento de la información, la base de datos redis, y el procesamiento de los estados de las placas enviado por los circuitos.
- **Frontend** es la capa de presentación del sistema, diseñada para que los usuarios interactúen de manera intuitiva con la aplicación, en este proyecto, corresponde a la proyección del mapa sobre la mesa, estos mapas se proyectaran a través del uso de Mapbox GL JS con los datos previamente obtenidos por el backend.

Para establecer la comunicación del backend con la base de datos y del backend con el frontend se hizo el uso de APIs, los cuales son mecanismos que permiten a dos componentes de software comunicarse entre sí mediante un conjunto de definiciones y protocolos. Para ello se usó Api websocket y Api restufl, las cuales nos permiten hacer o de información de forma full

duplex y semiduplex respectivamente; En el sistema se usará websocket para la comunicación del backend con frontend, y restful para la comunicación de la base de datos, y las placas de desarrollo con el backend[20].

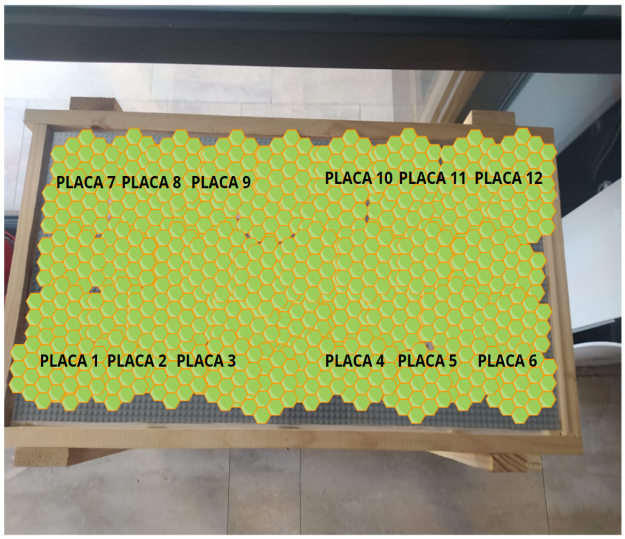
Para la programación de la API se usó **FastApi**, el cual es un framework de python, que nos permite programar Api de websocket y APISrestful, los cuales a través de endpoints, permite acceder o enviar información a través de métodos HTTP.[21]

Primero, para la realización de todos los cálculos correspondientes al backend, es fundamental tener claridad sobre los objetivos en función de los datos disponibles.

En la base de datos, por cada categoría de estudio se cuentan con dos mapas. Por ejemplo, en la categoría de trámites, se tienen los mapas **mapa-tramites:1** y **mapa-tramites:0**. Estos mapas representan los estados futuros y actuales del sector Costanera de Concepción y la accesibilidad a ese servicio en un escenario presente y uno futuro. Los datos correspondientes a estos mapas están en formato GeoJSON, como se explicó anteriormente, donde contienen polígonos con las mismas geometrías, pero con propiedades de "path-length"diferentes en algunos casos. El objetivo es utilizar los sensores RFID para detectar cada placa y determinar si corresponde a una placa actual o futura. Dependiendo de esta identificación, el backend del sistema proyectará únicamente en la zona de esa placa los polígonos y los colores correspondientes al estado actual o al estado futuro.

Para aclarar esta explicación, se presenta la Figura 3.17, que simula la proyección de los mapas en forma de colores. En esta representación, el color verde corresponde al mapa futuro, mientras que el rojo representa el mapa actual. Por otro lado, la Figura 3.31c ilustra el mapa a proyectar, el cual se construye a partir de los otros dos mapas y varía según el estado actual de cada placa.

(a) Simulación mapa actual



(b) Simulación mapa futuro.

(c) Simulación mapa a proyectar sobre mesa interactiva .

Fig. 3.17: Explicación mapa a proyectar, construido en base al mapa actual y futuro.

Una vez definida la idea, se procede a la construcción del backend del sistema; Como primer paso, se filtraron las geometrías correspondientes a cada placa en la mesa, tal como se explicó anteriormente, los polígonos de los mapas actuales y futuros comparten las mismas geometrías. Estas geometrías están asociadas a través de sus IDs, que fueron proporcionados por el equipo de ciencia de datos del laboratorio mediante archivos JSON denominados placa1.json, placa2.json, y así sucesivamente para cada placa. Luego, se filtraron las geometrías correspondientes a los IDs de cada placa utilizando la función **ExtraerGeometriasYIds**, que toma como entrada los

IDs proporcionados.

A continuación, para extraer los valores de path_length, que determinan el color de los polígonos, se creó la función **ExtraerPathLengths**. Como su nombre indica, esta función extrae la propiedad path_length del mapa seleccionado.

Finalmente, se encuentra la función **ActualizarPathLength**, que recibe como entradas el mapa al que se desea modificar la propiedad path_length, el resultado de la función **ExtraerPathLengths** (que puede provenir del mapa actual o futuro, según el estado del sensor: 1 para placas futuras y 0 para placas actuales), y las geometrías de la placa obtenidas mediante la función **ExtraerGeometriaseIds**.

Una vez aplicadas estas funciones, la propiedad path_length se actualiza únicamente en las geometrías correspondientes a la placa especificada en la función.

Con gran parte de la lógica del programa finalizada, se procede a la creación de la API y sus respectivos endpoints. Para probar los endpoints antes de la programación del circuito, se utilizó el software **Postman**, una herramienta de desarrollo que sirve para probar APIs, permitiendo el envío de solicitudes HTTP [22].

A continuación, se detallan los endpoints utilizados en la API, incluyendo una explicación de cada uno y su funcionalidad.

- **/ (Endpoint Raíz)**: Este endpoint se encarga de servir el archivo HTML principal, junto con sus archivos asociados de JavaScript y CSS, cada vez que se accede a la IP donde está desplegada la aplicación. Este archivo HTML contiene la aplicación de Mapbox GL JS.
- **/ws.** : Endpoint de websocket, encargado de crear la conexión websocket, aceptarla, y crear el canal de comunicación bidireccional para el envió de información hacia el frontend.
- **`/categoria/{categoria}`**: Este endpoint, que acepta el método GET, permite seleccionar la categoría del mapa a través de la URL. Utiliza la categoría especificada para realizar una consulta a la base de datos correspondiente y ejecutar la función **ExtraerPathLengths** sobre los mapas de esa categoría.
- **/process-json"**: Este endpoint, que acepta el método POST, maneja los estados enviados por los sensores de los circuitos 1 y 2. Al recibir los datos, los guarda temporalmente y los combina en un único JSON. Luego, recorre los canales de este JSON para consultar los valores de cada canal. Si el valor es 1, se ejecuta la función **ActualizarPathLength** en

la geometría correspondiente, intercambiando los valores con el mapa futuro. Si el valor es 0, la misma función se ejecuta intercambiando los valores con el mapa actual. Finalmente, el mapa resultante se envía al frontend a través del WebSocket.

Con todas las funciones y la APIs construidas, se ejecuta el siguiente comando: **uvicorn main –host 0.0.0.0 –port 8000 –reload**. Este comando inicia FastAPI, haciendo que la aplicación se este ejecutando en la IP del dispositivo en el puerto 8000. Además, al escuchar en todas las interfaces de red disponibles, permite que otros dispositivos, como las placas de desarrollo, se conecten y se comuniquen con la aplicación.

Para una mejor comprensión, nuestro backend está representado en el diagrama de flujo que se muestra en la Figura 3.18.

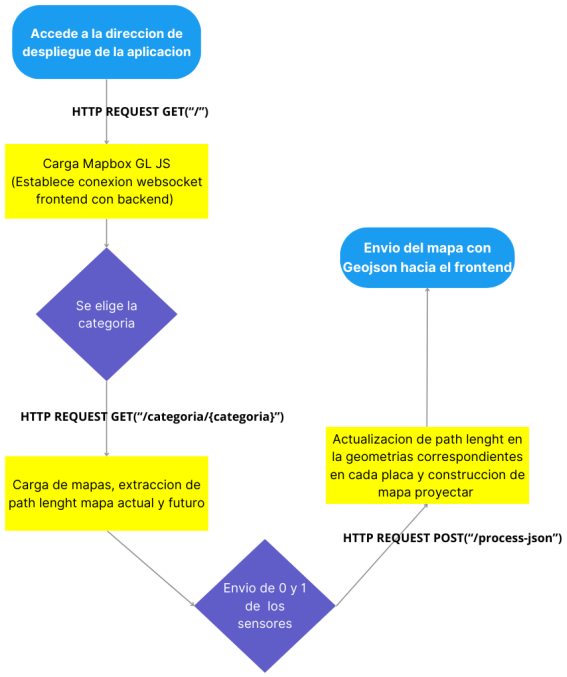


Fig. 3.18: Diagrama de flujo backend.

Con el backend completado, se procedio con el desarrollo del frontend. Tal como se mencionó anteriormente, se utilizará Mapbox GL JS para la aplicación web, lo que implicará el uso de HTML, JavaScript y CSS. Cada una de estas herramientas desempeña un papel específico en la aplicación, que se detalla a continuación.

- **HTML**:Es el lenguaje usado para la esctructuración de una pagina web, a través de distintos atributos, se van organizando los elementos de una pagina web.

- **Javascript**: es el lenguaje que permite darle interactividad y funcionalidad a la pagina web, como a los elementos que fueron creados en HTML previamente.
- **CSS**: Es el lenguaje encargado de la presentación visual de los elementos que fueron creados en HTML previamente, como colores, animaciones, etc..

Ya teniendo presentes las herramientas que se usaran para el desarrollo del frontend, se sigue la documentación de Mapbox para poder ejecutar la aplicación , y ver como poder proyectar los polígonos que van a ser enviados desde el backend en formato Geojson.[23]

Primero, en JavaScript, se debe establecer una conexión con el backend a través de WebSocket. Esto se lleva a cabo utilizando la función **WebSocket**, donde se especifican la IP del equipo y el puerto en el que se está ejecutando la API. Con el método **socket.onmessage**, se gestionan los datos que llegan desde el backend, que en este caso consisten en los datos GeoJSON de los polígonos del mapa a proyectar.

Tras inicializar Mapbox, se configura una fuente de datos vacía denominada **hexagonos source**, encargada de recibir los datos GeoJSON provenientes del WebSocket. Finalmente, se añade una capa al mapa para visualizar estos datos, extrayendo las propiedades de path_length y asignando rangos de colores que van del rojo al verde, los polígonos en rojo indican una mala accesibilidad a un servicio, mientras que los de color verde indican una buena accesibilidad.

Una vez creado el backend, se procede a probar el sistema utilizando Postman y desplegando la aplicación mediante el comando de FastAPI mencionado anteriormente.

Después de desplegar la aplicación, se accede a la IP correspondiente a través de un navegador, mostrando lo que aparece en la Figura 3.19.

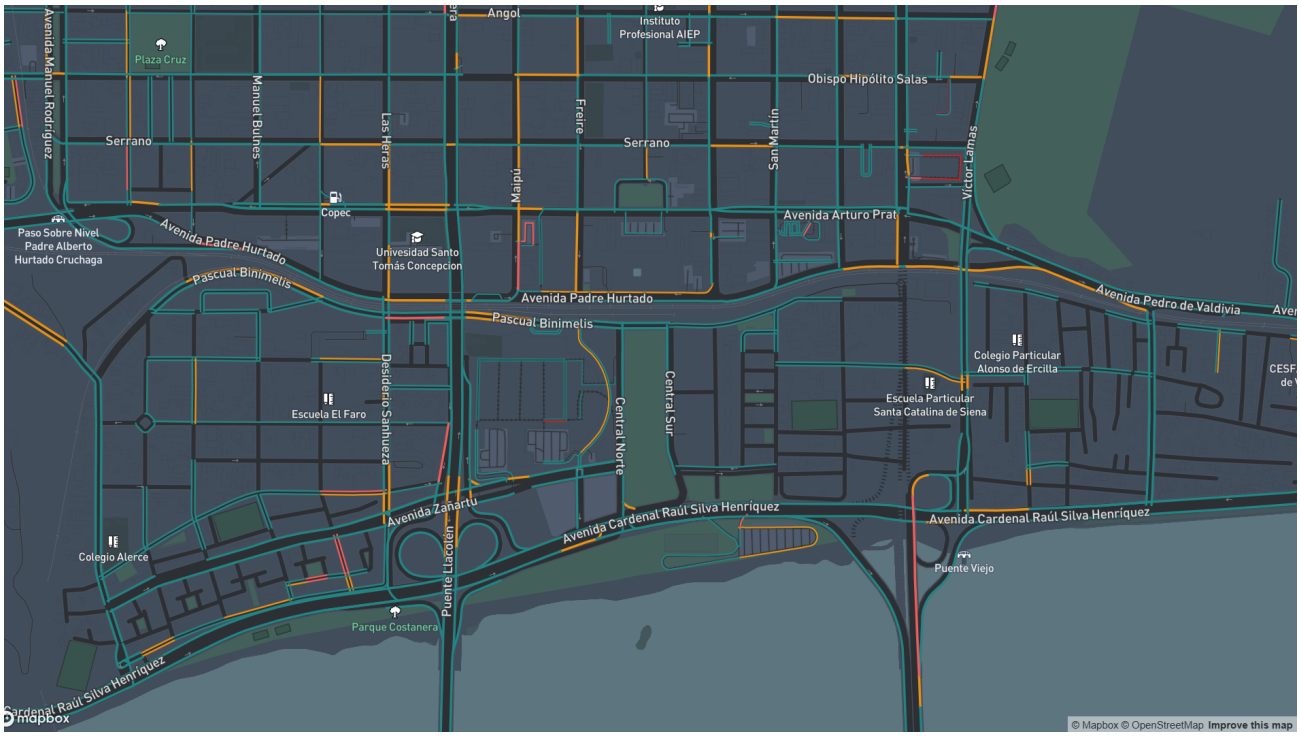


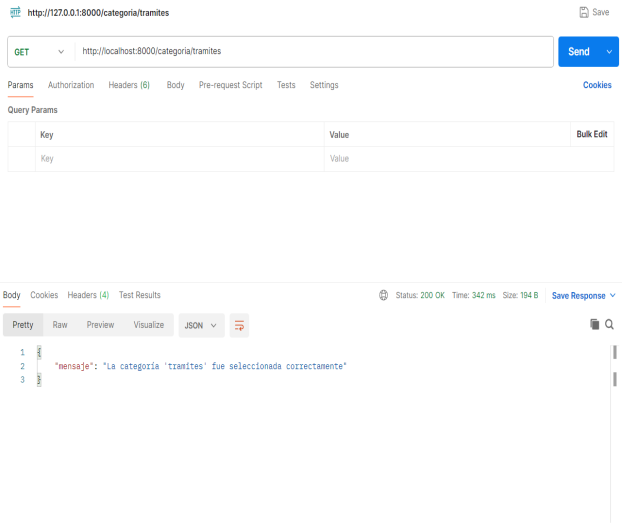
Fig. 3.19: Mapbox inciado y centrado en sector Costanera.

Una vez que Mapbox está en funcionamiento, se procedio a probar el sistema utilizando Postman, como se muestra en la Figura 3.20. Se realiza una solicitud HTTP GET a la URL:

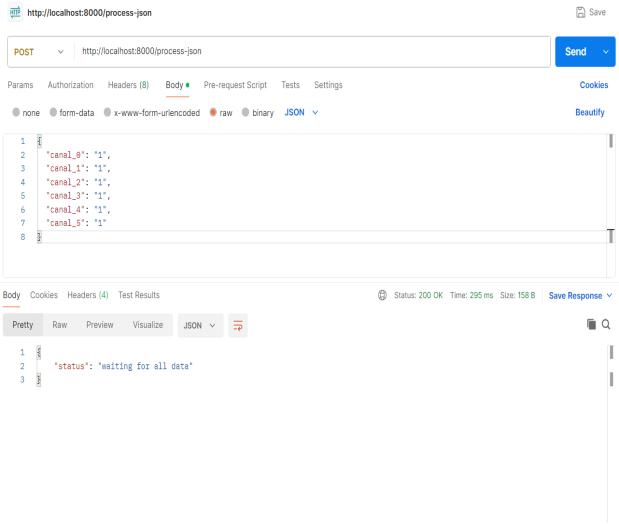
http://localhost:8000/categoria/tramites para cargar la categoría de trámites. A continuación, se envía una solicitud POST a la URL **http://localhost:8000/process-json**, incluyendo en el cuerpo de la solicitud los datos que deberían enviar las dos placas de desarrollo. Inicialmente, se analiza el mapa futuro configurando todos los canales para que envíen los siguientes valores:

```
1 "canal_0": "1", "canal_1": "1", "canal_2": "1", "canal_3": "1", "canal_4":
    "1", "canal_5": "1"
```

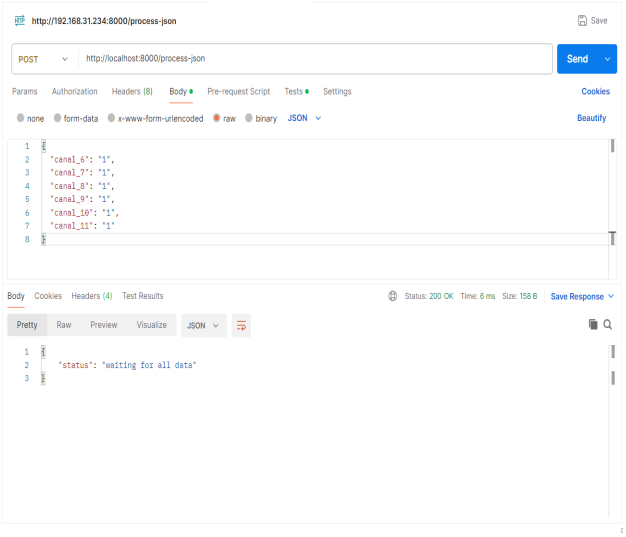
como se ilustra en la Figura 3.21.



(a) Get request para elegir categoría



(b) Post request con estados canales 0 al 5



(c) Post request con estados canales 6 al 11.

Fig. 3.20: Http requests enviadas por Postman.

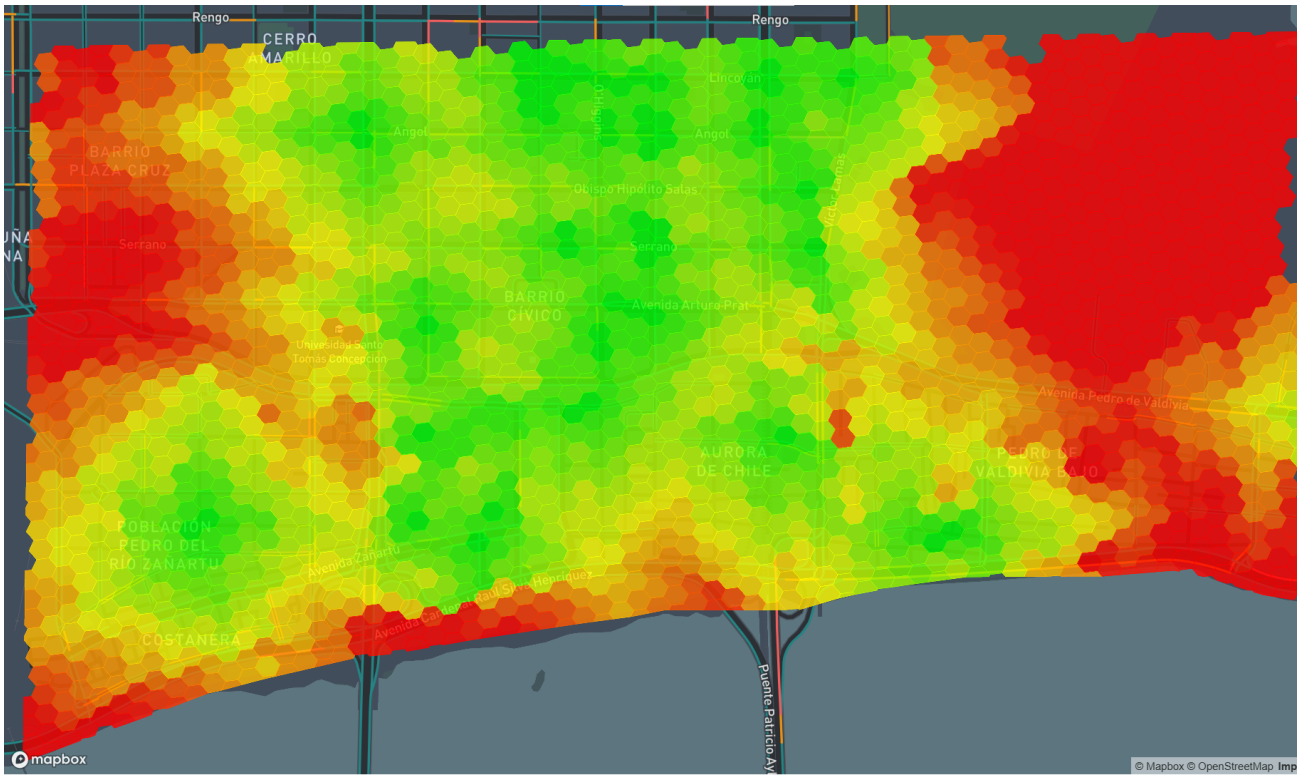


Fig. 3.21: Mapa futuro proyectado en Mapbox.

Luego, siguiendo un procedimiento similar al anterior, se prueba el sistema configurando todos los canales para que envíen un 0, con el fin de proyectar el mapa correspondiente al estado actual. Esta configuración genera la visualización mostrada en la Figura 3.22.

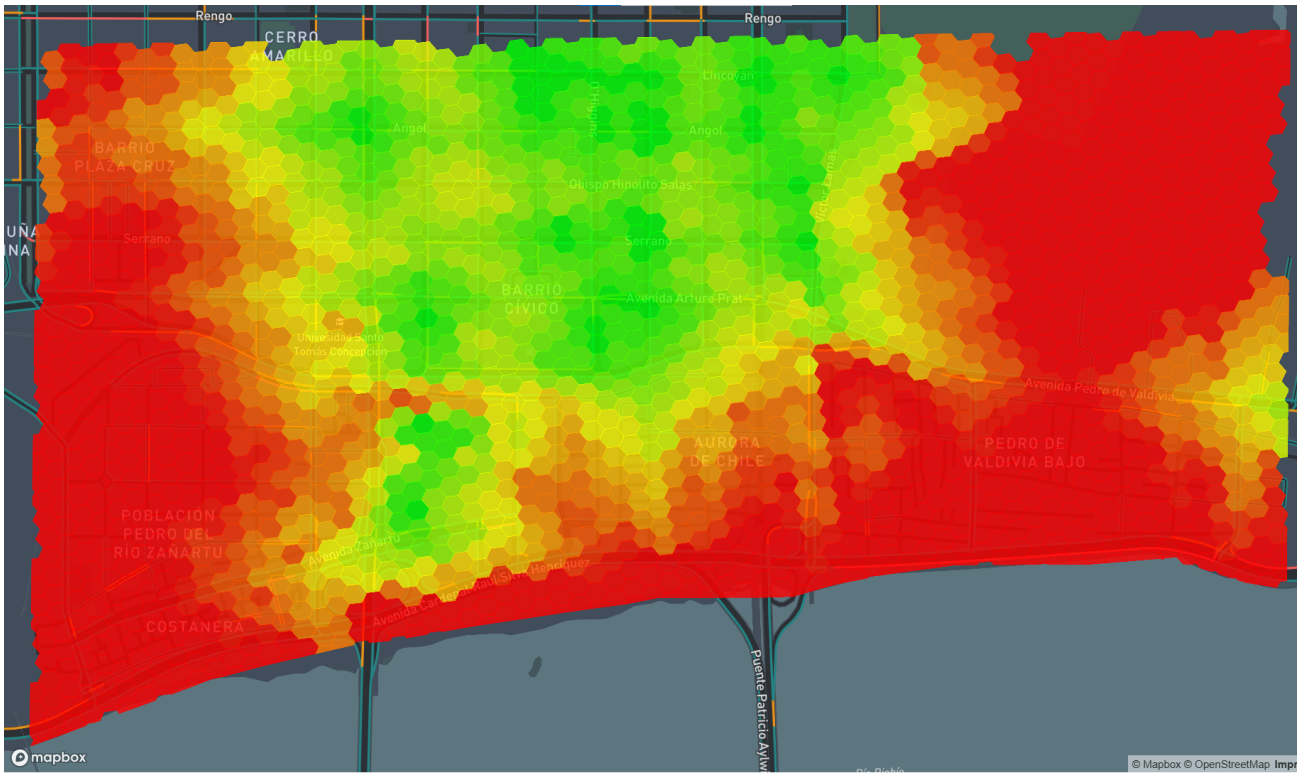


Fig. 3.22: Mapa actual proyectado en Mapbox.

Para probar la interactividad de las placas, se envío estados diferentes en cada uno de los canales, simulando los cambios en la mesa, a través de Postman, de manera análoga a lo realizado anteriormente. En este caso, se envio los siguientes valores:

```
"canal_0": "1", "canal_1": "0", "canal_2": "0", "canal_3": "1", "canal_4":
    "1", "canal_5": "1"
```

lo que nos da como resultado el mapa de le Figura 3.23.

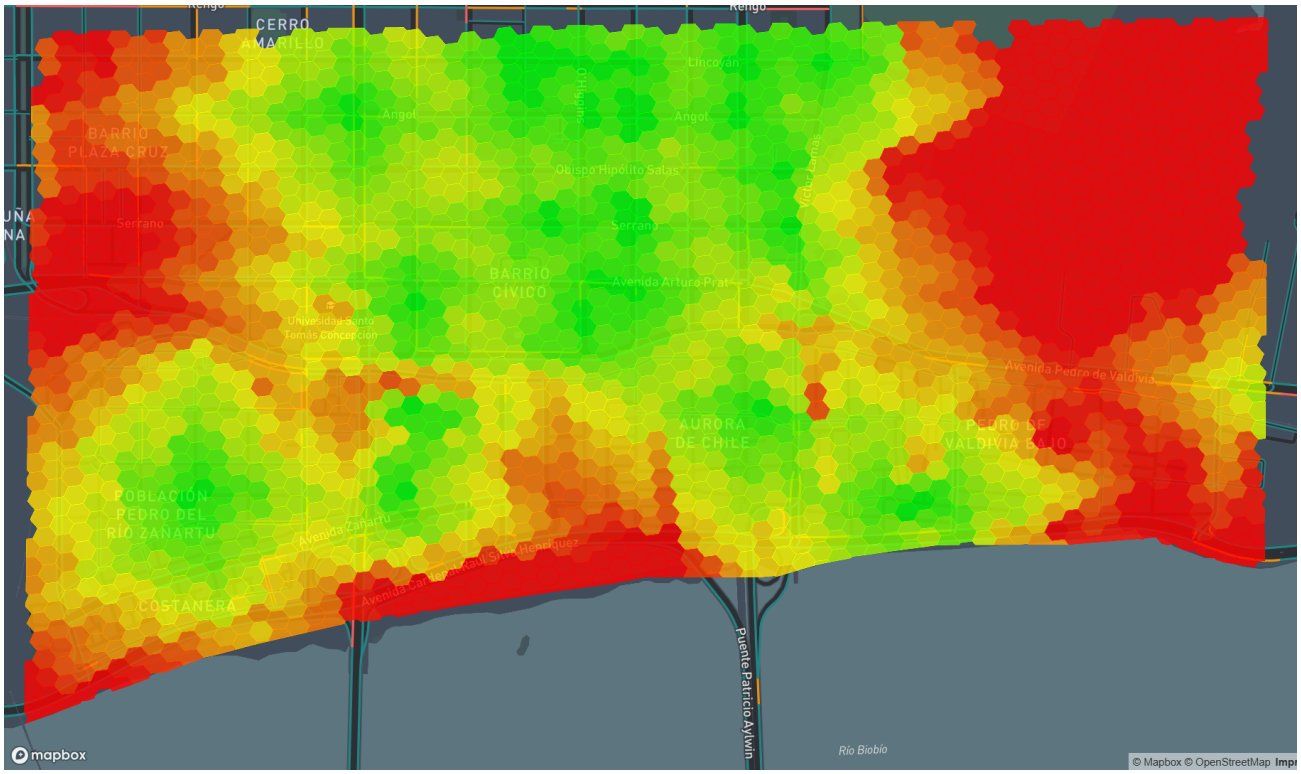


Fig. 3.23: Mapa combinado.

Lo cual prueba el correcto funcionamiento del sistema frente a los cambios que lleven hacia el backend.

Ya con todo el sistema desarrollado, el diagrama representativo de este se ve en la Figura 3.24.

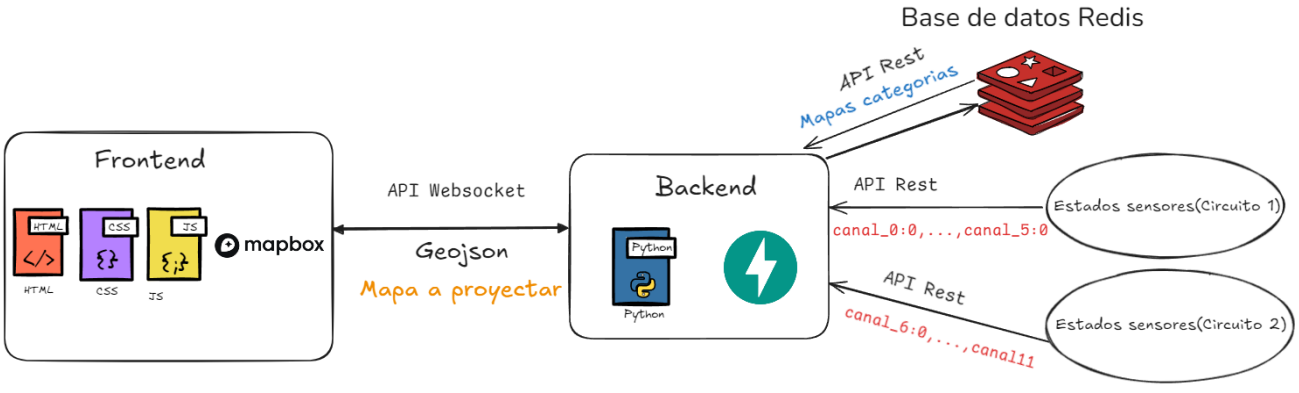


Fig. 3.24: Diagrama final del sistema.

3.6. Programación del circuito de sensores RFID.

Con el sistema de proyección en funcionamiento, el siguiente paso es desarrollar el programa destinado a las placas de desarrollo.

El objetivo principal será utilizar los sensores para detectar los UID ubicados en la parte inferior de las placas, asignarles un valor de 0 o 1, y almacenar esta información en formato JSON. Luego, los datos se enviarán al backend a través de la API previamente creada. Además, se integrarán las funcionalidades de la placa de desarrollo M5Stack, como su pantalla y botones, para implementar un menú que permita seleccionar la categoría que se desea proyectar en la mesa.

Con todos los objetivos establecidos, se procedió al desarrollo del programa utilizando Arduino IDE. Esta plataforma de desarrollo facilita la programación de microcontroladores y es compatible con las placas M5Stack empleadas en el proyecto. La estructura de programación en Arduino IDE está basada en el lenguaje C++.

En este programa, se implementaron diversas funciones con el objetivo de simplificar el proceso de detección de las placas y el envío de la información correspondiente. A continuación, se describen de manera secuencial las funciones implementadas:

- **Connecttowifi**:Función encargada de conectar la placa a la red Wi-Fi, utilizando el Access Point del teléfono celular. Además, muestra en la pantalla del M5Stack un mensaje confirmando la conexión exitosa, junto con la dirección IP asignada.
- **getUIDString**: Función encargada de poder leer el uid correspondiente de los tags que se acercan al sensor RFID del circuito y lo guarda en una variable.
- **MapUIDtovalue**: Función encargada de comparar el uid leido por el sensor, dentro de los uid enrolados correspondientes a cada placa, si es futuro se le asigna un 1, y si es presente, se le asigna un 0.
- **CheckRFIDonChannel**:Función encargada de ejecutar las dos funciones declaradas anteriormente, y guardar sus valores en un arreglo con su estado correspondiente en cada canal.
- **CreateJSON**:Función encargada de convertir el arreglo creado anteriormente, en formato JSON, para poder ser enviado posteriormente al backend.

- **SendPostRequest**: Función encargada de mandar el HTTP Post request al backend del sistema con el JSON con el estado de todos los canales en el.

Es importante destacar que todas las funciones, excepto ConnectToWiFi, se ubican en la sección del loop, ya que se ejecutan de manera iterativa sobre todos los canales. Debido a la naturaleza del proyecto, estas funciones deben estar en constante funcionamiento para garantizar una detección continua.

Para la creación del menú, se utilizó la librería **M5stackMenuSystem**, que facilita la creación de menús, la generación de ítems, y la interacción con ellos. Cada ítem corresponde a una categoría que se desea proyectar. Una vez seleccionada mediante los botones, se envía una solicitud HTTP GET a la API, permitiendo la selección de la categoría que se proyectará en el mapa. Esta interfaz se puede observar en la figura 3.25.

Fig. 3.25: Menú encargado de seleccionar la categoría del mapa a proyectar.

Todos los códigos correspondiente a este sistema se encuentra disponibles en el GitHub de Francisco Valdés.[24]

3.7. Automatización y despliegue de sistema en Raspberry pi.

Con el sistema ya desarrollado, es necesario implementar una forma de ejecutarlo en cualquier plataforma donde se desee realizar la proyección. Para este propósito, se ha decidido implementarlo en una Raspberry Pi, utilizando contenedores Docker.

Docker es una plataforma que permite a los desarrolladores crear, desplegar y ejecutar aplicaciones de manera aislada, garantizando que estas funcionen de la misma manera en cualquier entorno. Se basa en contenedores, los cuales son paquetes ligeros y portátiles que incluyen todo lo necesario para que una aplicación opere correctamente, como el código, las bibliotecas, las dependencias y las configuraciones. Esta funcionalidad asegura la portabilidad y consistencia entre distintos entornos de desarrollo, prueba y producción, eliminando problemas asociados con diferencias en las configuraciones del sistema [25].

Como primer paso, se crean dos archivos esenciales para el despliegue y la configuración de la aplicación: el archivo **docker-compose.yml** y el archivo **Dockerfile**. A continuación, se detalla la función de cada uno:

- **docker-compose**:Este archivo define y organiza múltiples servicios en contenedores Docker. Utilizando este archivo, se pueden declarar de manera sencilla todos los servicios necesarios.
- **dockerfile**:Dockerfile es un archivo que contiene una serie de instrucciones para construir la imagen Docker de nuestra aplicación.

Ya aclarado la función de estos archivos, el contenido del archivo Dockerfile, esta dado por la Figura 3.26.

```
# Usa una imagen base de Python
FROM python:3.11-slim

# Establece el directorio de trabajo
WORKDIR /app

# Copia el archivo de requerimientos al contenedor
COPY requirements.txt .

# Instala las dependencias
RUN pip install --no-cache-dir -r requirements.txt

# Copia el resto del código al contenedor
COPY . .

# Expone el puerto en el que corre FastAPI
EXPOSE 8000

# Comando para ejecutar la aplicación
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Fig. 3.26: Contenido archivo dockerfile.

Lo que hace este Dockerfile es establecer una imagen base de Python para el contenedor, crea un directorio de trabajo dentro del contenedor llamado /app, y luego copia el archivo requirements.txt, que contiene las bibliotecas y frameworks necesarios. Después, instala las

dependencias, copia el resto del código al contenedor, expone el puerto donde se ejecutará FastAPI y finalmente ejecuta el comando para desplegar la aplicación.

En cuanto al contenido del archivo docker-compose, esta dado por la Figura 3.27.

```yaml
version: '3.9'

services:
  api:
    build: .
    container_name: fastapi_app
    ports:
      - "8000:8000"
    depends_on:
      - redis
    environment:
      - REDIS_HOST=redis
      - REDIS_PORT=6379

  redis:
    image: "redis:alpine"
    container_name: "redis"
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data

volumes:
  redis_data:
```

Fig. 3.27: Contenido archivo docker-compose.

Lo que hace este archivo docker-compose es gestionar los siguientes servicios:

- **api**:Despliega la aplicacion fast api, busca el archivo dockerfile y ejecuta sus comandos, y se conecta con la base de datos redis, haciendo uso de sus variables de entorno.
- **redis**:Inicia una base de datos redis en el puerto 6379.

Finalmente, se copia todo el proyecto, incluidos los archivos de Docker, a la Raspberry Pi. Luego, desde la terminal, se ejecuta el comando **docker-compose up –build**, que se encarga de construir las imágenes y levantar los contenedores necesarios para ejecutar la aplicación.

3.8. Resultados

Con todo el sistema desarrollado y configurado, se realizaron las pruebas correspondientes en el laboratorio para verificar su funcionamiento. Primero, se creó un punto de acceso desde un teléfono móvil, al cual se conectaron todos los dispositivos, incluyendo la Raspberry Pi y las dos placas de desarrollo. Luego, se accedió remotamente a la Raspberry Pi y se procedió a ejecutar los contenedores en ella. Inicialmente, se puso en marcha la base de datos Redis, seguida de la copia de la base de datos del laboratorio hacia la base de datos Redis de la Raspberry Pi. Posteriormente, se inició el contenedor de FastAPI. Finalmente, se accedió a la dirección localhost:8000 para ejecutar Mapbox.

Como se observa en la Figura 3.28, el proyector se colocó en su trípode sobre una mesa, ajustando la imagen para que se proyectara de manera precisa sobre el modelo de la ciudad.

Fig. 3.28: Proyector desplegando Mapbox sobre la mesa.

A continuación, se selecciona la categoría que se desea visualizar a través del menú; para esta demostración, se eligió la categoría de trámites. Esto dio como resultado la proyección mostrada

en la Figura 3.29, donde todas las placas se encuentran en su estado actual.

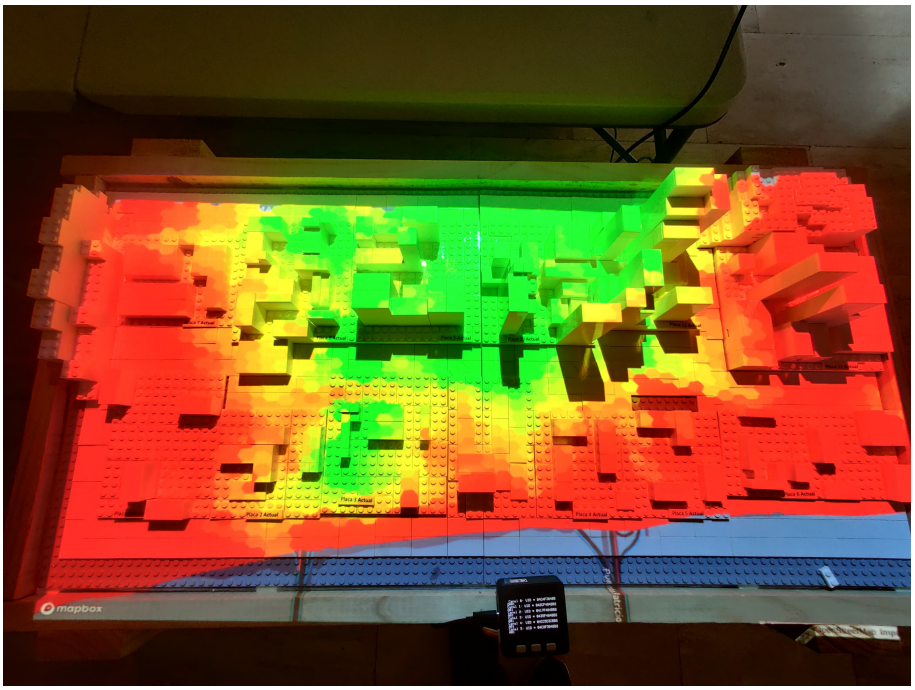
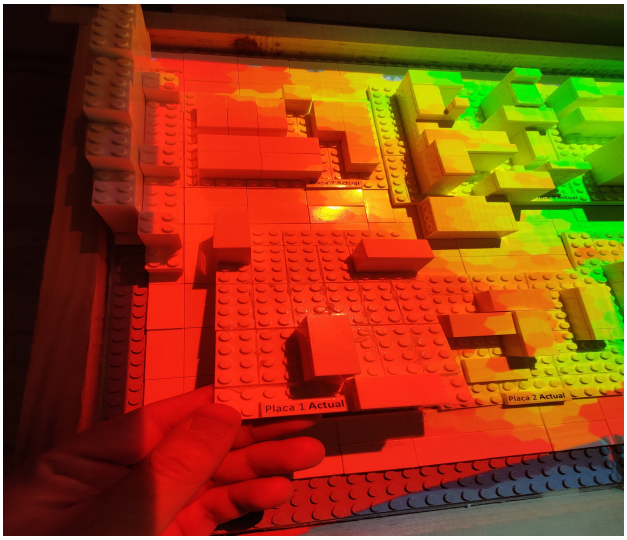
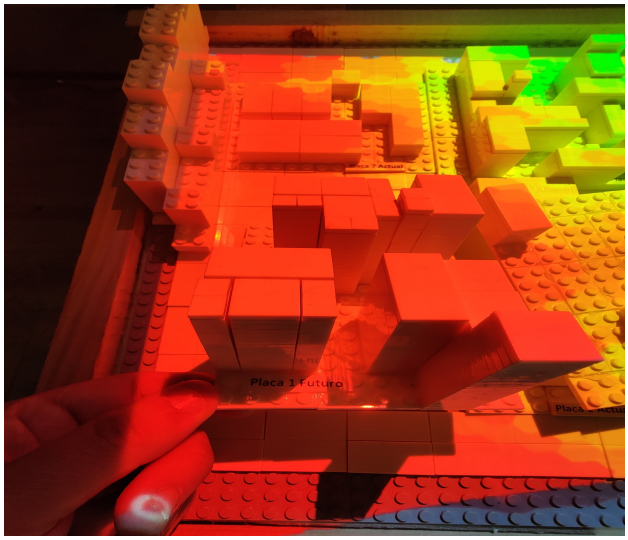


Fig. 3.29: Mapa de tramites en estado actual.

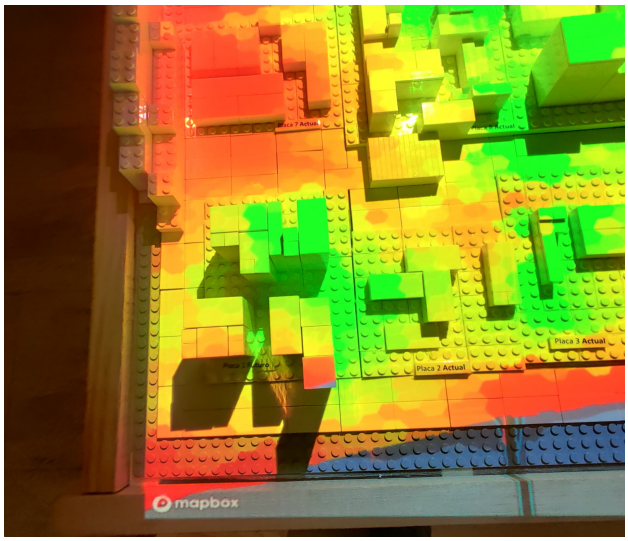
Una vez seleccionada la categoría, se probó la interactividad de las placas realizando un cambio en la placa 1, pasando de un estado actual a un estado futuro. Esto se ilustra en la Figura 3.30, donde se observa un cambio en los colores proyectados en las geometrías correspondientes a la placa 1, lo que indica que la transición entre el escenario actual y el futuro fue exitosa.



(a) Proyección placa 1 Actual



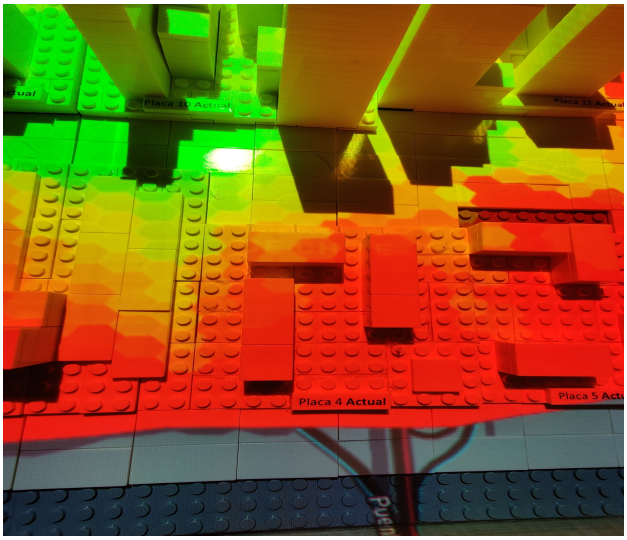
(b) Cambio de placa de 1 Actual a 1 Futuro



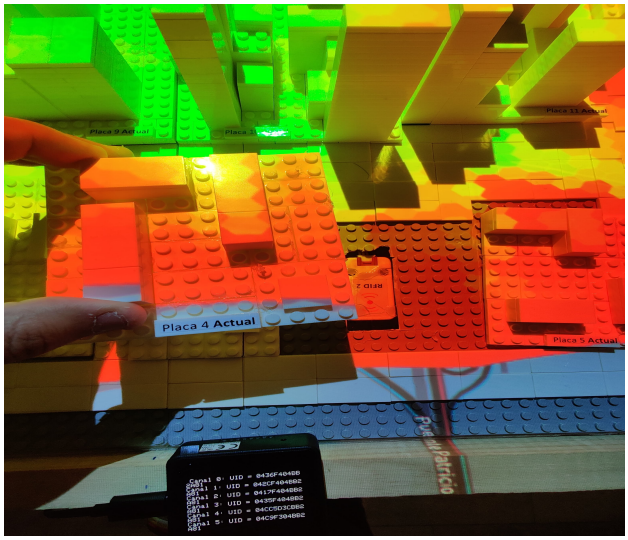
(c) Proyección placa 1 Futuro .

Fig. 3.30: Cambio de placa 1 Actual a placa 1 Futuro.

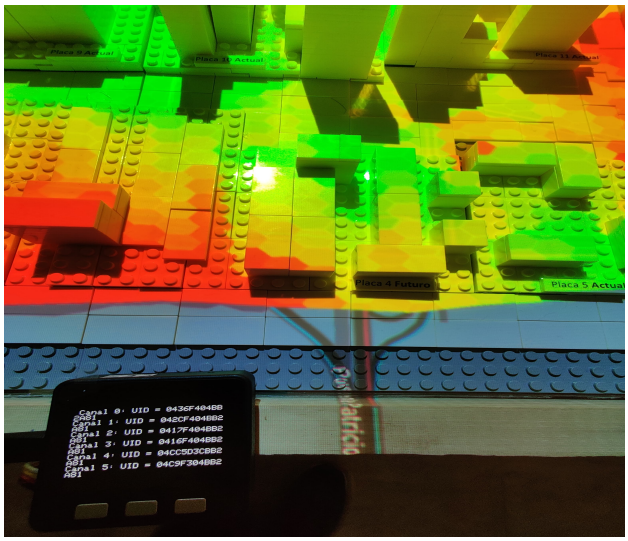
Una vez realizado el cambio en la placa 1 de un escenario actual a futuro, se procedió a realizar un cambio en la placa 4 para analizar su respuesta al tener diferentes estados. De manera análoga al ejemplo anterior, se observa en la Figura 3.31 cómo se proyecta el escenario actual en la placa 4. Luego, al realizar el cambio al estado futuro de la placa 4, se evidencia un cambio en los colores proyectados, lo que indica que la transición entre el escenario actual y el futuro fue exitosa.



(a) Proyección placa 4 Actual.



(b) Cambio de placa de 4 Actual a 4 Futuro.



(c) Proyección placa 4 Futuro.

Fig. 3.31: Cambio de placa 4 Actual a placa 4 Futuro.

Finalmente, se tiene la Figura 3.32, que muestra la proyección total de la mesa, incluyendo los cambios realizados en la placa 1 y la placa 4 hacia un escenario futuro.

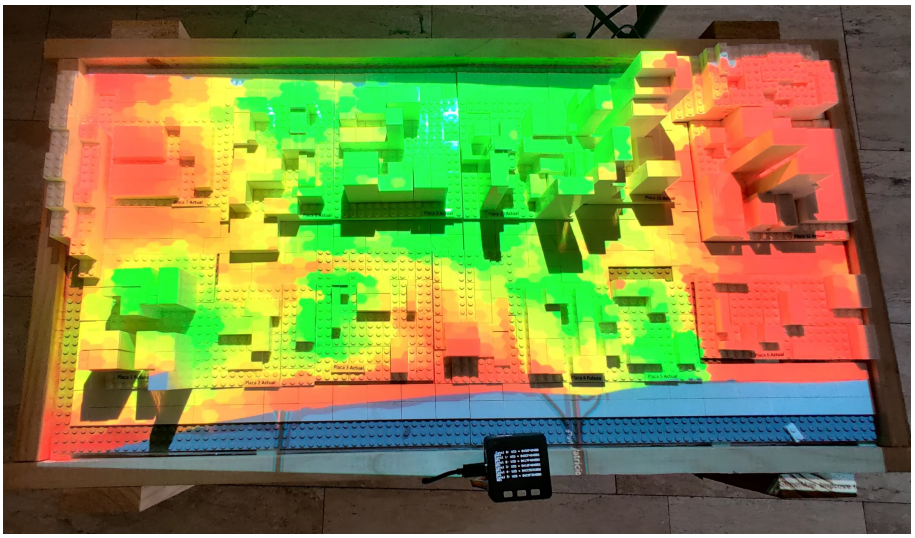


Fig. 3.32: Mapa de tramites con placa 1 y placa 4 en escenario futuro.

Ya con todo el sistema creado, y con sus pruebas pertinentes, se analizo su comparación con el sistema anterior mediante la tabla 3.3.

3.9. Tablas

Componente	**Cantidad**
Placa de desarrollo programable	2
Sensores RFID	14
Multiplexores	2
Raspberry pi	1
Proyector	1

Tabla 3.1: Tabla de componentes electrónicos

Componente	**Cantidad**	**Precio total**
Placa de desarrollo programable	2	$75,574
Sensores RFID	14	$32,829
Multiplexores	2	$15,058
Raspberry pi	1	$ 139,990

Tabla 3.2: Tabla de precios componentes.

Camaras	**RFID**
Sensibilidad frente a la luz	Fiabilidad frente a cambios de iluminación
Calibración de cámaras frente a movimientos de mesa	Versatilidad frente a movimientos de mesa
Uso base de datos imágenes	Uso base de datos con Mapbox
Datos fijos previamente cargados y establecidos	Versatilidad frente a cambios en mapa

Tabla 3.3: Tabla comparativa sistemas

4. Conclusión

Este proyecto ha abordado la necesidad de Citylab Biobío de contar con un cityscope portátil, así como también ha solucionado las fallas en el proceso de detección. Se ha desarrollado un nuevo sistema de detección, haciendo uso de sensores RFID, junto con su programa de detección y envió de información correspondiente, y se ha construido un backend actualizado, aprovechando las nuevas bases de datos y herramientas disponibles en el laboratorio.

En el futuro, se espera llevar el cityscope portátil a presentaciones adicionales. Dada la naturaleza del sistema, se anticipa que podrá seguir expandiéndose, incorporando nuevas funcionalidades, cargando datos adicionales y desarrollando nuevos modelos. Esto permitirá no solo realizar estudios en el sector Costanera de Concepción, sino también explorar diversos sectores de la zona.

Bibliografía

[1] Cideu,Concepción-Chile,(Año desconocido), extraído de https://www.cideu.org/miembro/concepcion-chile/.

[2] Citylab Bío Bío,Investigadores urbanos del MIT participan en inauguración de laboratorio de ciudad del Gran Concepción, 2023, extraído de https://citylabbiobio.cl/investigadores-urbanos-del-mit-participan-en-inauguracion-de-laboratorio-de-ciudad-d

[3] Citylab Bío Bío, Vecinos del Barrio Costanera experimentan por primera vez con CityScope, 2023, extraído de https://citylabbiobio.cl/vecinos-del-barrio-costanera-experimentan-por-primera-vez-con-cityscope/.

[4] MIT Media lab, Cityscopejs introduction, 2024, extraído de https://cityscope.media.mit.edu/cityscopejs/Introduction/.

[5] A. Grignard et al., CityScope Hanoi: interactive simulation for water management in the Bac Hung Hai irrigation system, 2020 12th International Conference on Knowledge and Systems Engineering (KSE), Can Tho, Vietnam, 2020.

[6] OpenCV Docs,Detection of ArUco Markers,(Año desconocido), extraído de https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html.

[7] DipoleRFID, What is RFID, (Año desconocido), extraído de https://www.dipolerfid.com/rfid-blog/what-is-rfid.

[8] Rafael Jiménez Bravo, "Sistema de seguimiento de objetos usando OpenCv, ArUco y Filtro de Kalman extendido", Proyecto Fin de Grado, Ingeniería Electrónica, Robótica y Mecatrónica, 2018, Departamento de Ingeniería de Sistemas y Automática, Escuela Técnica Superior de Ingeniería, Universidad de Sevilla.

[9] Manuals.plus, Guía del usuario del marcador de biblioteca ArUco, 2023, extraído de https://manuals.plus/es/aruco/library-marker-manual

[10] H. C. Kam, Y. K. Yu and K. H. Wong, An Improvement on ArUco Marker for Pose Tracking Using Kalman Filter, 2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Busan, Korea (South), 2018.

[11] B. Li, J. Wu, X. Tan and B. Wang, 'ArUco Marker Detection under Occlusion Using Convolutional Neural Network' 2020 5th International Conference on Automation, Control and Robotics Engineering (CACRE), Dalian, China, 2020.

[12] M5Stack, About us, 2024 , extraído de https://m5stack.com/about-us

[13] M5Stack, ESP32 Basic Core loT Development Kit V2.7 , 2024 , extraído de https://shop.m5stack.com/products/esp32-basic-core-lot-development-kit-v2-7

[14] M5Stack, RFID 2 Unit , 2024, extraido de https://shop.m5stack.com/products/rfid-unit-2-ws1850s

[15] M5Stack, I2C Hub 1 to 6 Expansion Unit , 2024 , extraído de https://shop.m5stack.com/products/i2c-hub-1-to-6-expansion-unit-pca9548apw

[16] Citylab Bío Bío, City Lab Biobío y Municipalidad San Pedro de la Paz firman convenio para anticipar efectos de proyectos urbanos en movilidad, 2024 , extraído de https://citylabbiobio.cl/citylabbiobio-convenio-sanpedro/

[17] OpenStreetMaps,About us, 2024, extraído de https://www.openstreetmap.org/about

[18] Redis, Estructuras de datos, 2024 , extraído dehttps://redis.io/es/redis-enterprise/estructuras-de-datos/.

[19] Mapbox GL JS , Guides, 2024 , extraído de https://docs.mapbox.com/mapbox-gl-js/guides

[20] Amazon aws, ¿Qué es una interfaz de programación de aplicaciones (API)?, 2024 , extraido de https://aws.amazon.com/what-is/api/

[21] FastAPI, FastAPI framework, 2024, extraído de https://fastapi.tiangolo.com/

[22] Postman, What is Postman? , 2024 , extraído de https://www.postman.com/product/what-is-postman/

[23] Mapbox docs, Add a polygon to a map using a GeoJSON source, 2024 , extraído de https://docs.mapbox.com/mapbox-gl-js/example/geojson-polygon/

[24] Github, Repositorios Francisco Valdes, 2024, extraido de https://github.com/Fvaldesrojas?tab=repositories

[25] Docker, what-container,2024, extraido de https://www.docker.com/resources/what-container/