UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingeniería

Monitoreo Remoto de Navegación de Vehículo no Tripulado Mediante Aplicación Android.

André Sebastián Pazmiño Harnisth
Ingeniería Electrónica

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
Ingeniero Electrónico

Quito, 24 de abril de 2022

Universidad San Francisco de Quito USFQ

Colegio de Ciencias e Ingeniería

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Monitoreo Remoto de Navegación de Vehículo no Tripulado mediante aplicación Android

André Sebastián Pazmiño Harnisth

Nombre del profesor, Título académico **René Játiva Espinoza, Ph.D.**

Quito, 24 de abril de 2022

© DERECHOS DE AUTOR

Nombres y apellidos: André Sebastián Pazmiño Harnisth

Código: 00134463

Cédula de identidad: 1722449707

Lugar y fecha: Quito, 24 de abril de 2022

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

Este proyecto se enfoca en la realización de un prototipo de aplicación de monitoreo remoto de navegación para un vehículo no tripulado automatizado. Usando la interfaz gráfica Android que se ha desarrollado, se aprecia información del estado de los distintos sensores emplazados sobre el vehículo de prueba. En esta etapa del proyecto se visualiza información de los sensores de proximidad, del rastreo georreferenciado, así como de datos almacenados concernientes a las trayectorias descritas por el vehículo.

Se realiza el procesamiento de datos en un Raspberry Pi, programado en Python. El Raspberry adquiere los datos de los sensores disponibles para el manejo del vehículo. Entonces, usando la base de datos que se ha implementado, esta información se almacenará en servidores externos y podrá transmitirse a la aplicación de monitoreo desarrollada en Android, a través de canales de comunicación. En la aplicación de Android se visualizará la información transmitida por los sensores de proximidad ylos datos transmitidos por el módulo GPS. Además, se almacenará las trayectorias del vehículo mientras la aplicación está en ejecución.

Palabras claves: Vehículo autónomo, GPS, Sensores de proximidad, Python, Raspberry Pi, Android.

ABSTRACT

This project focuses on the realization of a remote navigation monitoring application prototype for an automated unmanned vehicle. With the Android graphical interface that has been developed, information on the status of the different sensors located on the test vehicle can be seen. At this stage of the project, information from proximity sensors, georeferenced tracking, as well as stored data concerning the trajectories described by the vehicle, is displayed.

Data processing is performed on a Raspberry Pi, programmed in Python. The Raspberry acquires the data from the available sensors for vehicle management. Then, using the database that has been implemented, this information will be stored in external servers and can be transmitted to the monitoring application developed on Android, through communication channels. The information transmitted by the proximity sensors and the GPS module will be displayed in the Android application. In addition, the vehicle trajectory will be stores while the application is running.

Keywords: Autonomous vehicle, GPS, Proximity sensors, Python, Raspberry Pi, Android.

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

OBJETIVOS

- Diseño e implementación de un prototipo de aplicación, en plataforma Android, para el monitoreo del estado de navegación de un vehículo no tripulado.
- Provisión de capacidades de geoposicionamiento a un vehículo autónomo, incorporando un módulo GPS NEO 6M, creando y poniendo a punto para el efecto, un canal de comunicación entre el procesador Raspberry del vehículo y el módulo GPS.
- Realización de pruebas de latencia y funcionamiento de los sensores de proximidad implementados en el vehículo no tripulado.
- Realización de pruebas de operación y funcionamiento del módulo GPS.

INTRODUCCIÓN

Como en todas las industrias, con la llegada de nuevas tecnologías y la constante demanda del mercado de mejores productos, la industria automotriz ha tenido varias modificaciones. Comenzamos por los vehículos de motores de combustión con velocidades limitadas, pero para la época una importante revolución, frente a la tecnología alcanzada en la actualidad, con máquinas capaces de llegar a los 250 km/h. Pero no dejamos de pensar en el mañana y en las necesidades que debemos suplir con cada cambio que realizamos en la industria, continuamos en la búsqueda de ideas innovadoras que transformen el paradigma de la funcionalidad del automóvil y de lo que este es capaz de hacer.

El desafío al cual nos estamos enfrentando en la actualidad recae en las preguntas: ¿Cómo les entregamos mayor autonomía a los vehículos? Y además ¿cómo podremos disminuir a corto y largo plazo la contaminación que su constante uso produce en nuestro medio ambiente? Ambas problemáticas las estamos resolviendo hoy con un cambio completo del tipo de motor y sistema que los automóviles sueles utilizar, gracias a las compañías pioneras como Tesla, BYD, BAIC, entre otros, sabemos que es posible hacerlo mediante la fabricación del automóvil 100 % eléctrico, el cual tiene un nuevo mundo de posibilidades, donde somos capaces de empezar a probar su autonomía y modificar procesos para que no deje una huella dañina en el ecosistema.

En la Universidad San Francisco de Quito estamos probando este tipo de tecnología a pequeña escala, usando como modelo un vehículo eléctrico diseñado originalmente por su fabricante como un juguete para niños y que se ha modificado en el tiempo por los estudiantes de titulación de la carrera de Ingeniería Electrónica. El objetivo

del trabajo realizado en los últimos semestres por diferentes equipos, ha sido mejorar el diseño de forma que el carrito pueda moverse de forma automática, esquivando obstáculos y siguiendo una trayectoria previamente especificada. Sin embargo, el carrito también puede conducirse a control remoto. Presenta por tanto dos modos de conducción: control remoto usando un Control inalámbrico Dualshock 4 y modo autónomo a partir de los sensores implementados, que permiten que el programa que corre sobre un ordenador Raspberry Pi tome las acciones pertinentes sobre los motores eléctricos que controlan la trayectoria del vehículo. El presente trabajo complementa el realizado hasta el momento.

Una problemática en la actual implementación del vehículo automatizado es la dificultad de hacer evidente el estado de los sensores existentes y su interrelación con el sistema de procesamiento del vehículo. Por ese motivo se aprecia la necesidad de crear una interfaz de monitoreo que permita al desarrollador de software, observar y darse cuenta de lo que ocurre externamente y de cómo los sensores están percibiendo el entorno del vehículo cuando éste se encuentra en movimiento. En la actual implementación del vehículo, se trata de disponer de una mejor orientación de lo que pasa tanto hacia adelante como hacia atrás del carrito. El trabajo se enfoca en el diseño e implementación de un centro de monitoreo desarrollado inicialmente en la plataforma de Python para pasarla después a una aplicación móvil para Android.

El centro de monitoreo implementado muestra los datos de distancia que se obtienen de los sensores de proximidad, y realiza el rastreo del movimiento del vehículo usando un módulo GPS.

MARCO TEÓRICO

Unidad de procesamiento:

La unidad de procesamiento que se utilizará para el proyecto es un raspberry pi 4B.

Figura 1. Raspberry pi 4B.

Raspberry Pi, como se muestra en la figura 1, es un dispositivo electrónico que se comenzó a producir a partir del 2012 con la finalidad de tener un ordenador económico y práctico para el desarrollo educativo de aplicaciones. Cuenta con un sistema operativo llamado Raspbian, el cual ha sido desarrollado en base al sistema operativo de Linux. (Educación 3.0, 2018)

Dado que este ordenador es muy versátil para el desarrollo de proyectos, se ha realizado la automatización y monitoreo de un vehículo no tripulado. Para el caso de este proyecto se utiliza Python y se aplicarán sensores que sean compatibles con la unidad de procesamiento.

Módulo GPS NEO 6M:

El módulo de GPS NEO 6M, es un dispositivo electrónico que se utiliza como receptor de señales y por medio de distintas interfaces de comunicación transmite esta información en distintos procesadores, como son Arduino y Raspberry.

Figura 2. Módulo GPS NEO 6M.

Este equipo electrónico brinda sus señales mediante el protocolo NMEA que es un protocolo de tipo input/ output mediante el cual se podrá determinar la ubicación del vehículo en la superficie terrestre.

El módulo dispone de un sistema de comunicación serial de tipo TTL (Transistor-Transistor- Logic), esto indica que el voltaje de operación del módulo es con respecto al voltaje de polarización VCC de los transistores. Para el uso que se le va a emplear la lógica del puerto serial será de 3.3 V, a pesar de que el módulo, trabaja entre 3.3V y 5V. Si se quisiera trabajar con lógica de 5V se debe realizar un divisor de tensión que reduzca ese voltaje a 3.3V. (Clavijo Rujel, 2022)

Sensor de proximidad HC SR04:

La figura 3, muestra un sensor HC SR04 el cual utiliza ondas de sonido para su funcionamiento. Las ondas son transmitidas a una frecuencia de 40 Khz, que es una frecuencia mucho mayor a la frecuencia auditiva que aprecia el ser humano que va desde los 20 Hz hasta los 20 KHz, por este motivo no se escuchan las ondas emitidas por el sensor. (Criollo Merino, 2018)

Figura 3. Sensor de proximidad Ultrasónico HC SR04.

Lo que hace este sensor es transmitir una onda y cuando hay un obstáculo presente esta onda se refleja y llega al receptor de onda del sensor y mediante ecuaciones matemáticas se adapta esta señal a las especificaciones requeridas. Por lo tanto, la ecuación de adaptación que se utiliza es:

$$Distancia\ al\ objeto = \frac{Duración\ del\ pulso * velocidad\ de\ la\ luz}{2}$$

Android:

Android es un sistema operativo diseñado a partir del kernel 2.6 de Linux cuyo objetivo de diseño es para equipos móviles con pantalla táctil. Este sistema operativo servirá para diseñar la interfaz gráfica que utilizará el usuario para obtener una interacción de tipo input/output con los sensores del vehículo. En la figura 4 se puede apreciar que mediante una entrada del usuario se podrá ver la información de los sensores y del gps; luego la aplicación mediante la información de salida de la Raspberry, obtenida de los sensores mostrará los datos procesados. (Condori Calizaya, 2022)

Figura 4. Gráfico de Interacción del usuario con la aplicación de Android.

El lenguaje de programación que se implementa en la creación de la aplicación de Android es JavaScript y se lo puede desarrollar en Visual Studio Code y las pruebas se realizan mediante Android Studio, con el emulador de celular que tiene dicho software.

Base de datos:

La base de datos es un componente importante en la realización del proyecto, debido a que es nuestro fichero de almacenamiento de información que se utiliza para transferir datos desde la Raspberry del vehículo no tripulado hacia la aplicación móvil del usuario.

La base de datos que se utiliza en el proyecto es Firebase, de Google. Para el presente proyecto se utiliza una base de datos en tiempo real, en donde, los sensores, tanto el GPS como los sensores ultrasónicos de proximidad envían datos a la base de datos en tiempo real y esta base de datos los almacena y transporta esa información a la aplicación de Android diseñada. Por lo tanto, para poder realizar el monitoreo del vehículo se requiere que ambos dispositivos, tanto el vehículo como el dispositivo de monitoreo se encuentren conectados a una red de internet.

Python:

Los lenguajes de programación han evolucionado con el pasar de los años, partiendo de leguajes de programación de bajo nivel, en donde, las instrucciones se escriben en código binario; hasta lenguajes de programación de alto nivel.

Python es un lenguaje de programación de alto nivel, en donde, las instrucciones transmitidas se compilan mientras el programa se está ejecutando lo cual ralentiza el desempeño a diferencia de un lenguaje en donde durante la ejecución no existe interpretación. Pero lo que hace a este lenguaje un lenguaje muy utilizado en la industria debido a su accesibilidad y facilidad de uso al momento de programar, ya que es un lenguaje de programación gratuito y multiplataformas. (Sanner, 1975)

Android Studio:

Android Studio es un software creado para el desarrollo y diseño de aplicaciones móviles que cuentan con el sistema operativo de Android, cuenta con herramientas de emulación rápidas y cargadas de funciones de un dispositivo móvil mediante el cual se pueden realizar pruebas de funcionamiento de la aplicación desarrollada. Esta aplicación es libre para todos los usuarios que quieran crear su aplicación.

Para este proyecto se utiliza Android Studio únicamente para el uso del emulador de celular, ya que la programación y el desarrollo de la aplicación se lo desarrolla con JavaScript en la plataforma Virtual Studio Code

Javascript:

Según Eguíluz (2009), Javascript es un lenguaje de programación que se utiliza para la realización de páginas web dinámicas, en la cual se incorpora efectos y acciones que se activan al momento en que el usuario pulsa botones o crea un input hacia la pantalla del móvil. Es un lenguaje interpretado, con el cual se trabajará para que el usuario seleccione las opciones de monitoreo a las cuales quiera acceder.

DESARROLLO DEL PROYECTO

Lista de materiales utilizados para la elaboración:

1. **Vehículo no tripulado BMW i8 Spyder Toy Rollplay:**

El vehículo en el cual se implementará el proyecto es el vehículo autónomo desarrollado por los estudiantes del semestre pasado de ingeniería electrónica. Para este vehículo se hicieron ciertas modificaciones, introduciendo circuitería eléctrica para el control del vehículo por medio de un microprocesador, entre otros dispositivos electrónicos, como motores eléctricos, reguladores de voltaje, baterías, etc. Se mantuvo las luces, el sistema mecánico y de tracción del vehículo.

Durante el procedimiento se presentaron inconvenientes con el sistema eléctrico del vehículo, por ese motivo las pruebas se vieron limitadas a la accesibilidad de manejo que brindaba el vehículo.

2. **Raspberry Pi:**

La Raspberry Pi, es el sistema de procesamiento del vehículo en el cual se conectan los sensores y el módulo gps, con los que se realiza la transmisión de datos para la aplicación remota externa del vehículo. Esto se puede lograr gracias a que la raspberry tiene puertos GPIO, los cuales sirven para establecer los canales de comunicación entre los sensores y el ordenador.

3. **Sensores de proximidad HC-SR04:**

El vehículo cuenta con 4 sensores de proximidad: uno en la parte frontal, uno a la izquierda, uno a la derecha y uno en la parte posterior. Mediante estos sensores se puede procesar el manejo automático del vehículo.

Para este proyecto se utiliza únicamente la conducción vía control remoto del vehículo para verificar la transmisión de los estados de los sensores.

4. **Módulo GPS Neo 6M:**

El módulo GPS se implementa en el vehículo para posibilitar su rastreo mientras se encuentra en movimiento. Esta aplicación encuentra un inconveniente ya que al momento de establecer conexión serial mediante el GPS con el sistema de procesamiento, se desconecta la conexión bluetooth del mando de ps4, lo que ocasiona que se pierda esa funcionabilidad de manejar el vehículo sin necesidad de usar cable para establecer conexión entre el mando y el vehículo. La solución que se propone para este problema es conectar un adaptador para crear otro puerto de comunicación UART, para de esta manera poder tener uso simultáneo del bluetooth y del módulo GPS.

Etapas de desarrollo del proyecto

1. **Esquema de conexión de aplicación de monitoreo remoto:**

El esquema de conexión que se presentará a continuación es un esquema simplificado, donde se muestra únicamente la conexión de los sensores que se utilizaron para el proyecto. Estos sensores son:

- Módulo GPS NEO 6M.
- 4 sensores de proximidad HC-SR04

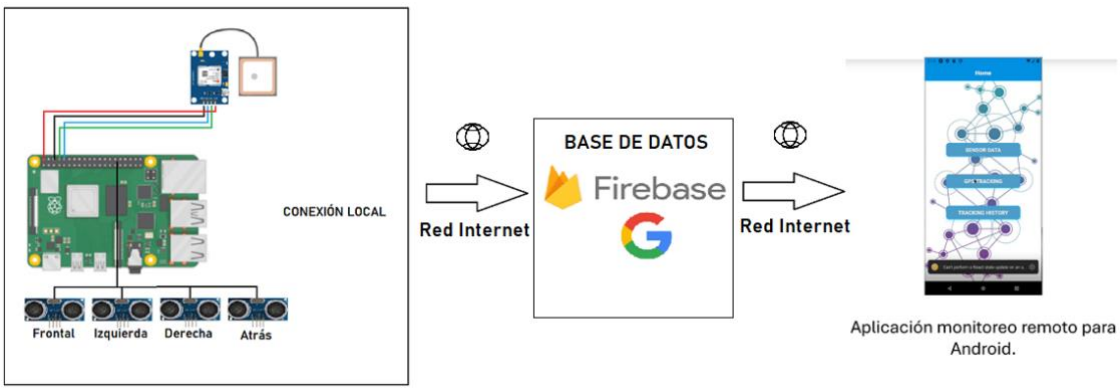

Figura 5. Diagrama de conexión de aplicación de monitoreo remoto.

2. **Conexión de GPS NEO 6M a la raspberry:**

Uno de los enfoques principales del proyecto es la implementación de un sistema de localización y rastreo para el vehículo autónomo no tripulado. Por lo tanto, para esta aplicación se debe utilizar el módulo GPS NEO 6M, el cual es compatible con el sistema de procesamiento con el cual se trabaja.

De esta forma, se realiza una conexión serial, por la terminal UART de la raspberry; mediante los siguientes pasos:

1. Primero se debe conectar los pines del módulo GND, VCC, RX, TX, con los GPIO correspondientes a la conexión UART de la raspberry. Por lo tanto, Rx se conecta con GPIO15; el Tx con el GPIO14, VCC con la fuente de poder de 3.3V y GND con cualquiera de los pines GND que queden disponibles. De esta manera queda establecida la conexión física del dispositivo.
2. Una vez realizada la primera la fase de conexión física del módulo GPS con la raspberry, se debe activar y crear el canal de comunicación entre los dos dispositivos. Para este procedimiento, se enciende la raspberry y se realizan los siguientes procesos:
    - Se abre el centro de comandos de la Raspbian y se inserta el código mostrado en la figura 6 que permite ingresar a la configuración de los puertos de comunicación de la raspberry:

```
sudo nano /boot/config.txt
```

Figura 6. Código de acceso a la configuración de puertos de Raspbian, ingresado en el centro de comandos

    - Una vez que se ingresa al centro de configuración de puertos, se debe modificar los puertos, deshabilitando el puerto bluetooth de la raspberry, como se muestra en la figura 7:

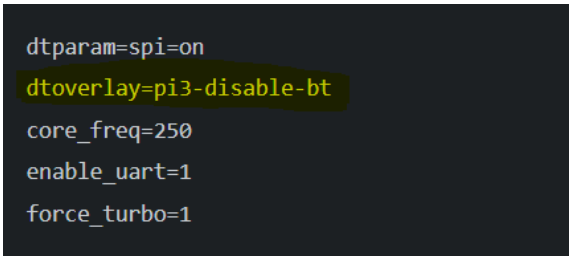

Figura 7. Modificación de los canales de comunicación de la raspberry, donde se deshabilita el bluetooth.

La razón principal por la que se necesita deshabilitar bluetooth mientras se usa el módulo GPS es que ambas usan el puerto ttyAMA0. Este ttyAMA0 puede ser los pines Tx, Rx para Raspberry Pi. Pero para Raspberry Pi 3 y versiones más nuevas, se usa para Bluetooth. Ahora, para obtener datos del módulo GPS, se necesita que el pin tx del módulo se conecte con el pin rx de Raspberry Pi. Básicamente, se obtendrá los datos GPS del módulo a través de una conexión en serie. Ahora bien, para esta comunicación serial se necesita el puerto ttyAMA0, que el bluetooth usa por defecto. Entonces, cuando se necesita usar el puerto ttyAMA0, se debe deshabilitar el bluetooth.

- Una vez que se realiza esta modificación se reinicia la raspberry y posteriormente, cuando los cambios se actualizan se escribe el siguiente comando, en el centro de comandos:

```
sudo cat /dev/ttyAMA0
```

Figura 8. Comando para verificar que la comunicación con el GPS.

Este comando se utiliza para verificar el protocolo GPS NMEA-0183, el cual es un protocolo de transmisión de datos de ubicación que envía el GPS.

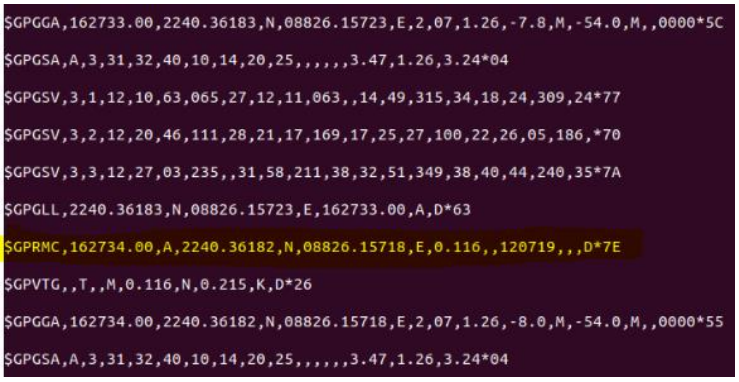

Figura 9. Información recibida mediante protocolo NMEA-0183.

El protocolo NMEA- 0183, es un protocolo de comunicación serial que está basado en código ASCII, el cual se compone por oraciones, lo que permite que sea comprensible al momento de requerir la información. En la figura 9 se ha resaltado la oración GPRMC, que se utiliza para el desarrollo del proyecto porque es la pertinente para manejar información de posicionamiento. Esta oración tiene los siguientes campos:

Campo	**Estructura**	**Descripción**
1	$GPRMC	Encabezado del registro.
2	UTC	Tiempo en el cual se registra el dato.
3	Estado Pos.	Estado de la posición: A= Datos válidos, V= Datos no válidos.
4	Latitud	Latitud.
5	Dirección Lat	Dirección de Latitud: Norte y Sur.
6	Longitud	Longitud.
7	Dirección Lon	Dirección de Longitud: Este u Oeste.
8	Velocidad Kn	Velocidad de movimiento sobre el suelo. U= nudos
9	Track True	
10	Fecha	Día, Mes y Año.

11	Variación Mag.	Variación mágnetica en grados, siempre será positiva.
12	Dirección Var.	Dirección de variación magnética. Variación del Este se resta del rumbo verdadero. Variación del Oeste se suma al rumbo verdadero.
13	Indicador de modo	El indicador de modo de posicionamiento se debe consultar en una tabla de modo del sistema de posicionamiento NMEA.

Tabla 1. Descripción de campos para el encabezado de registro $GPRMC.

De estos campos que se muestran en la tabla 1, se usan únicamente los campos 4 y 6, que son los campos de los cuales obtendremos las coordenadas georreferenciales para la localización del vehículo usando la aplicación de monitoreo.

Una vez realizados estos procedimientos se concluye con la conexión y establecimiento del canal de comunicación entre la raspberry y el módulo GPS.

3. Diseño e implementación de centro de monitoreo local:

Para el diseño del centro de monitoreo local, que se muestra en la figura 10, con la finalidad de explicar el funcionamiento y como se traslada información de los sensores a la aplicación de monitoreo.

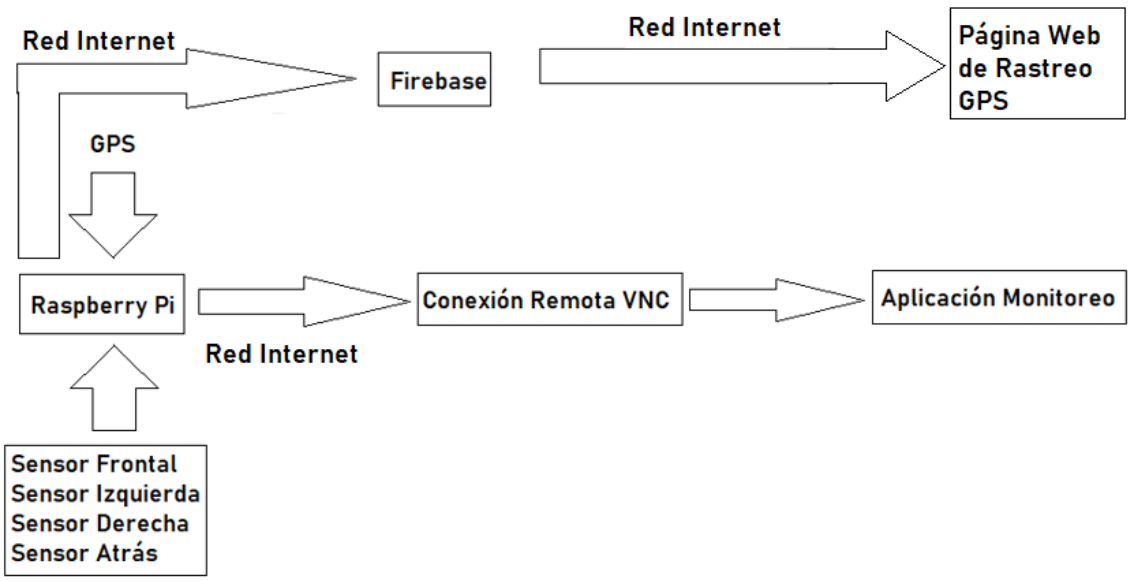

Figura 10. Diagrama de Bloques Funcionamiento Centro Monitoreo Local.

Para el centro de monitoreo que se aprecia en la figura 10 se utiliza únicamente una Raspberry como dispositivo de procesamiento de datos, debido a que dicho ordenador debe mantenerse conectada a internet para poder enviar los datos a la pagina web que muestra el rastreo georreferencial realizado por el GPS.

En versiones anteriores se utilizaba la Raspberry como access point para poder conectar otra Raspberry, para realizar la operación esclavo/maestro, pero al momento de crear un Access Point con la Raspberry, se enfoca el proyecto a estar ligado en una red local y eso limita la exportación de datos a la web la cual requiere salida de datos de la Raspberry y mediante el uso del internet como canal de comunicación poder utilizar los datos del GPS e implementarlos con la API de Google maps, la cual se utiliza como herramienta para obtener la geolocalización del vehículo..

Una vez que se conoce el diagrama de funcionamiento de la aplicación de monitoreo local se procede a realizar el diseño de la interfaz gráfica para el usuario, para la cual se utilizaron como herramientas varias librerías, entre las cuales están:

- Pynmea2: Esta librería permite la interacción entre el GPS y la aplicación que se va a diseñar. Por medio de esta librería se obtendrán las variables de latitud y longitud que se utilizarán para la aplicación de monitoreo local.
- Pyrebase: Esta librería es para exportar e importar datos de una base de datos hacia la aplicación que se quiera programar. Para la aplicación de este proyecto únicamente se realizará la exportación de datos.
- Pygame: Esta librería se implementa para el diseño de la interfaz gráfica para el usuario.

A continuación, se muestra el código, mediante el cual se realiza la programación para el diseño de la aplicación de monitoreo local:

```
import pygame
import sys
from pygame.constants import USEREVENT
import pygame.locals
import random
import webbrowser
import serial
import time
import string
import pynmea2
import threading
import RPi.GPIO as GPIO
import pyrebase

firebaseConfig={
"apiKey": "AIzaSyCfWGfOvfmv7aV74H5UzTjeNZZjMbfDfTU",

    "authDomain": "gps-tracker-710.firebaseapp.com",

    "projectId": "gps-tracker-710",

    "storageBucket": "gps-tracker-710.appspot.com",

    "messagingSenderId": "529606423358",

    "appId": "1:529606423358:web:2148437f69544c74ebb26f",

    "measurementId": "G-VQ7VYJVQ0W",
    "databaseURL": "https://gps-tracker-710-default-rtdb.firebaseio.com"
    }

firebase=pyrebase.initialize_app(firebaseConfig)
db=firebase.database()

GPIO.setmode(GPIO.BCM)

gt1 = 23       # gt1 ge1 CON SENSOR FRONTAL
ge1 = 24

gt2 = 17       #gt2 ge2 CON SENSOR IZQUIERDO
ge2 = 27
```

```
gt3 = 9         ##gt3 ge3 CON SENSOR DERECHO
ge3 = 11

gt4 = 5         #gt4 ge4 CON SENSOR POSTERIOR
ge4 = 6

distance_list= [888,888,888,888]
dis_thresh = 5.00

GPIO.setup(gt1, GPIO.OUT)  #gt1 ge1 CON SENSOR FRONTAL
GPIO.setup(ge1, GPIO.IN)
GPIO.setup(gt2, GPIO.OUT)  #gt2 ge2 CON SENSOR IZQUIERDO
GPIO.setup(ge2, GPIO.IN)
GPIO.setup(gt3, GPIO.OUT)  ##gt3 ge3 CON SENSOR DERECHO
GPIO.setup(ge3, GPIO.IN)
GPIO.setup(gt4, GPIO.OUT)  #gt4 ge4 CON SENSOR POSTERIOR
GPIO.setup(ge4, GPIO.IN)



lat = 0
lng = 0

WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
GREEN = (0, 255, 0)
RED = (255, 0, 0)
BLUE = (0, 0, 255)

T1 = "Distancia Sensores:"
T2 = "Sensor Frontal:  "
T3 = "Sensor Izquierda:  "
T4 = "Sensor Derecha:  "
T5 = "Sensor Posterior:  "
T6 = "GPS Rastreo Movimiento"
T7 = "LAT:  "
T8 = "LNG:  "

pygameDisplay = pygame.display.set_mode((800, 600), 0, 32)


class Main_Rect(object):
```

```
class Main_Rect(object):
    def __init__(self):
        pygame.init()
        self.font = pygame.font.SysFont('Arial', 25)
        self.screen = pygame.display.set_mode((800, 600), 0, 32)
        self.screen.fill(WHITE)
        pygame.display.update()

    def addRect(self):
        self.rect = pygame.draw.rect(self.screen, BLACK, (5, 5, 790, 590), 2)
        pygame.display.update()


class Upper_Rect(object):
    def __init__(self):
        pygame.init()
        self.font = pygame.font.SysFont('Arial', 25)
        self.screen = pygame.display.set_mode((800, 600), 0, 32)
        self.screen.fill(WHITE)
        pygame.display.update()

    def addRect(self):
        self.rect = pygame.draw.rect(self.screen, BLACK, (5, 25, 790, 570), 2)
        pygame.display.update()


class Lower_Rect(object):
    def __init__(self):
        pygame.init()
        self.font = pygame.font.SysFont('Arial', 25)
        self.screen = pygame.display.set_mode((800, 600), 0, 32)
        self.screen.fill(WHITE)
        pygame.display.update()

    def addRect(self):
        self.rect = pygame.draw.rect(self.screen, BLACK, (5, 320, 790, 275), 2)
        pygame.display.update()

    def addText(self):
        self.screen.blit(self.font.render('CAMARAS', True, BLACK), (350, 320))
        pygame.display.update()
```

```
class Two_Rect_Left(object):
    def __init__(self):
        pygame.init()
        self.font = pygame.font.SysFont('Calibre', 25)
        self.screen = pygame.display.set_mode((800, 600), 0, 32)
        self.screen.fill(WHITE)
        pygame.display.update()

    def addRect(self):
        self.rect = pygame.draw.rect(self.screen, BLACK, (5, 25, 395, 295), 2)
        pygame.display.update()

    def addText(self):
        pygame.time.set_timer(pygame.USEREVENT, 1000)
        distancia1 = distance_list[0]
        distancia2 = distance_list[1]
        distancia3 = distance_list[2]
        distancia4 = distance_list[3]

        self.screen.blit(self.font.render(T1, True, BLACK), (50, 30))
        self.screen.blit(self.font.render(T2 + str(distancia1), True, BLACK), (50, 55))
        self.screen.blit(self.font.render(T3 + str(distancia2), True, BLACK), (50, 75))
        self.screen.blit(self.font.render(T4 + str(distancia3), True, BLACK), (50, 95))
        self.screen.blit(self.font.render(T5 + str(distancia4), True, BLACK), (50, 115))
        if distancia1 < dis_thresh:
            pygame.draw.circle(self.screen, RED, (200, 170), 30, 0)
            pygame.draw.circle(self.screen, BLACK, (200, 170), 32, 2)
        else:
            pygame.draw.circle(self.screen, GREEN, (200, 170), 30, 0)
            pygame.draw.circle(self.screen, BLACK, (200, 170), 32, 2)
        if distancia2 < dis_thresh:
            pygame.draw.circle(self.screen, RED, (115, 210), 30, 0)
            pygame.draw.circle(self.screen, BLACK, (115, 210), 32, 2)
        else:
            pygame.draw.circle(self.screen, GREEN, (115, 210), 30, 0)
            pygame.draw.circle(self.screen, BLACK, (115, 210), 32, 2)
        if distancia3 < dis_thresh:
            pygame.draw.circle(self.screen, RED, (285, 210), 30, 0)
            pygame.draw.circle(self.screen, BLACK, (285, 210), 32, 2)
        else:
            pygame.draw.circle(self.screen, GREEN, (285, 210), 30, 0)
            pygame.draw.circle(self.screen, BLACK, (285, 210), 32, 2)

        if distancia4 < dis_thresh:
            pygame.draw.circle(self.screen, RED, (200, 270), 30, 0)
            pygame.draw.circle(self.screen, BLACK, (200, 270), 32, 2)
        else:
            pygame.draw.circle(self.screen, GREEN, (200, 270), 30, 0)
            pygame.draw.circle(self.screen, BLACK, (200, 270), 32, 2)
        pygame.display.update()


def gps():
    global lat
    global lng
    while True:
        port = "/dev/ttyAMA0"
        ser = serial.Serial(port, baudrate=9600, timeout=0.5)
        dataout = pynmea2.NMEAStreamReader()
        newdata = ser.readline()
        n_data = newdata.decode('latin-1')
        if n_data[0:6] == '$GPRMC':
            newmsg = pynmea2.parse(n_data)
            lat = newmsg.latitude
            lng = newmsg.longitude
            gps = "Latitude=" + str(lat) + " and Longitude=" + str(lng
            data = {"LAT": lat, "LNG": lng}
            db.update(data)
            print(gps)


def distancia(gt, ge, n):
    while True:
        global distance_list
        GPIO.output(gt, True)
        time.sleep(0.00001)
        GPIO.output(gt, False)
        StartTime = time.time()
        StopTime = time.time()

        while GPIO.input(ge) == 0:
            StartTime = time.time()

        while GPIO.input(ge) == 1:
```

```
        while GPIO.input(ge) == 1:
            StopTime = time.time()

        TimeElapsed = StopTime - StartTime
        distance = (TimeElapsed * 34300) / 2
        distance_list[n]= distance

class Two_Rect_Right(object):
    def __init__(self):
        pygame.init()
        self.font = pygame.font.SysFont('Calibre', 25)
        self.screen = pygame.display.set_mode((800, 600), 0, 32)
        self.screen.fill(WHITE)
        pygame.display.update()

    def addRect(self):
        self.rect = pygame.draw.rect(self.screen, BLACK, (400, 25, 395, 295), 2)
        pygame.display.update()

    def addText(self):
        pygame.time.set_timer(pygame.USEREVENT, 1000)
        self.screen.blit(self.font.render(T6, True, BLACK), (450, 30))
        self.screen.blit(self.font.render(T7 + str(lat), True, BLACK), (450, 55))
        self.screen.blit(self.font.render(T8 + str(lng), True, BLACK), (450, 75))
        pygame.display.update()

    def addButton(self):

        self.rectButton = pygame.draw.rect(self.screen, BLACK, (480, 150, 170, 50), 2)
        self.screen.blit(self.font.render('CLIC PARA ACCEDER', True, BLACK), (500, 170))
        pygame.display.update()

    def buttonClick(self):
        self.file = 'gps_without_cred.html'
        if self.rectButton.collidepoint(pygame.mouse.get_pos()):

            webbrowser.open(self.file, new=2)


if __name__ == '__main__':
    t1 = threading.Thread(target=gps)
    t1.start()
    t2 = threading.Thread(target=distancia, args=(gt1, ge1, 0))
    t2.start()
    t3 = threading.Thread(target=distancia, args=(gt2, ge2, 1))
    t3.start()
    t4 = threading.Thread(target=distancia, args=(gt3, ge3, 2))
    t4.start()
    t5 = threading.Thread(target=distancia, args=(gt4, ge4, 3))
    t5.start()
    MR = Main_Rect()
    UR = Upper_Rect()
    LR = Lower_Rect()
    TRL = Two_Rect_Left()
    TRR = Two_Rect_Right()
    MR.addRect()
    UR.addRect()
    LR.addRect()
    LR.addText()
    TRL.addRect()
    TRL.addText()
    TRR.addRect()
    TRR.addText()
    TRR.addButton()
    while True:
        for event in pygame.event.get():
            if event.type == USEREVENT:
                pygame.draw.rect(TRR.screen, WHITE, pygame.Rect(402, 27, 390, 100))
                pygame.draw.rect(TRL.screen, WHITE, pygame.Rect(7, 27, 390, 110))
                TRL.addText()
                TRR.addText()
            if event.type == pygame.MOUSEBUTTONDOWN:
                TRR.buttonClick()

            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
    t1.join()
    t2.join()
    t3.join()
    t4.join()
    t5.join()
```

Como resultado del código de programación se obtiene una aplicación de esta forma.

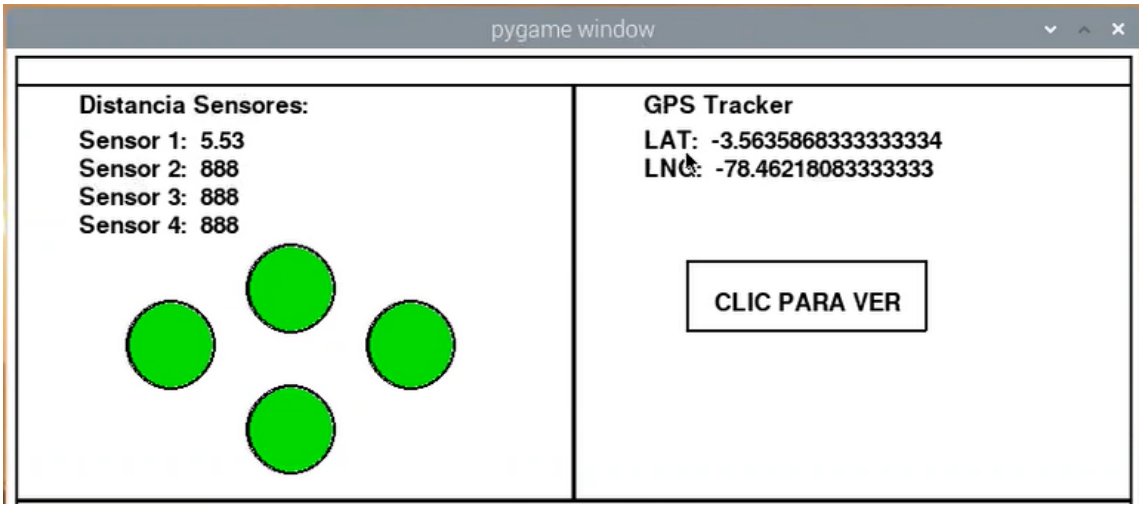

Figura 11. Aplicación de Monitoreo local Vehículo no tripulado.

El usuario observa en la aplicación los datos de la distancia, en centímetros, que reciben los sensores, acompañados con unos círculos dispuestos de tal manera que el usuario reconozca cuál de los sensores capta un objeto a menor distancia que el umbral especificado. Por otro lado, se obtienen también los datos de geolocalización proporcionados por el GPS. Se muestran la longitud y la latitud obtenidas del protocolo NMEA, acompañadas con un botón, que traslada al usuario a una página web que muestra sobre un mapa la ubicación y la ruta que va tomando el módulo GPS.

A continuación, se muestran los resultados:

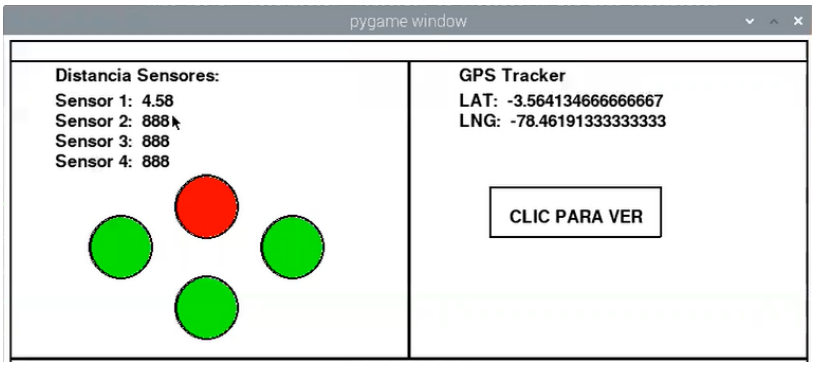

Figura 12. Resultados de variación de luz cuando existe un objeto al frente.

En el ejemplo de la Figura 12, el umbral especificado de la distancia hacia objetos cercanos para el cambio de color del círculo que representa al sensor se fijó en 5.00 cm. Como el sensor frontal ha determinado la presencia de un obstáculo a 4,78 cm, el círculo ha tomado el color rojo en lugar de verde. Este cambio alerta al usuario de la aplicación de un peligro inminente y en caso de conducción automática de que el carrito se encuentra realizando acciones para evitar el objeto.

Por otra parte, la figura 13 muestra el resultado de la ubicación obtenida por las coordenadas transmitidas y en la figura 14 se aprecia como la página web también permite observar la ruta que lleva el GPS:

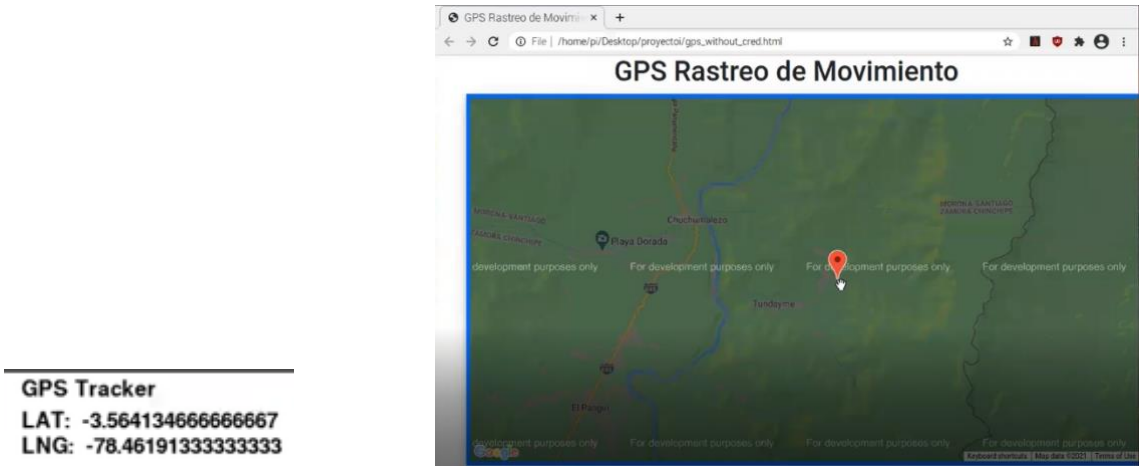

Figura 13. Leyenda de longitud y latitud exportada y ubicación en página web.

Figura 14. Rastreo de movimiento del módulo GPS.

Para este resultado es muy importante mostrar las variables de longitud y latitud que se recolectan en la aplicación de monitoreo del módulo GPS, que se muestran en la figura 13, para comparar con la ubicación mostrada en la página web de georreferencia de la figura 14. Para poder comparar los resultados obtenidos por nuestra página web, se insertó manualmente las coordenadas de longitud y latitud de la figura 13 y se determinó que la ubicación de la figura 14 concuerda con la ubicación señalada en la aplicación de monitoreo de la figura 13.

Un elemento importante para el entendimiento de cómo se trasladaron las variables a la base de datos es la explicación de cómo se obtienen dichos datos.

```
firebaseConfig={"apiKey":"AIzaSyCfWGfOvfmv7aV74H5UzTjeNZZjMbfDfTU" ,

    "authDomain": "gps-tracker-710.firebaseapp.com",

    "projectId": "gps-tracker-710",

    "storageBucket": "gps-tracker-710.appspot.com",

    "messagingSenderId": "529606423358" ,

    "appId": "1:529606423358:web:2148437f69544c74ebb26f",

    "measurementId": "G-VQ7VYJVQ0W",
    "databaseURL": "https://gps-tracker-710-default-rtdb.firebaseio.com"
    }

firebase=pyrebase.initialize_app(firebaseConfig)
db=firebase.database()

while True:
        port="/dev/ttyAMA0"
        ser=serial.Serial(port, baudrate=9600, timeout=0.5)
        dataout = pynmea2.NMEAStreamReader()
        newdata=ser.readline()
        n_data = newdata.decode('latin-1')
        if n_data[0:6] == '$GPRMC':
                newmsg=pynmea2.parse(n_data)
                lat=newmsg.latitude
                lng=newmsg.longitude
                gps = "Latitude=" + str(lat) + " and Longitude=" + str(lng)
                print(gps)
                data = {"LAT": lat, "LNG": lng}
                db.update(data)
                print("Data sent")
```

Figura 15. Código implementación base de datos.

Esta parte del código que se mostró previamente es importante explicar porque mediante esta parte del código se realiza el enlace de comunicación entre el programa del vehículo con la base de datos en tiempo real que se utilizará para transportar datos. En esta parte, lo que se realiza es activar el puerto ttyAMA0, mediante la variable port; se declara la base de datos, mediante la variable firebaseConfig, en donde se pegan las credenciales de la base de datos creada para la transportación; después, mediante el uso de pynmea se recolecta los campos 4 y 6 de la oración GPRMC del protocolo que son los campos de latitud y longitud; finalmente para exportar los datos desde la raspberry a la base de datos se implementa el código db.update (data).

Por último, para la implementación de la página web, se diseñó y desarrollo mediante programación html, en la cual se colocó un título a la página, un recuadro

en donde se abrirá la API de Google maps y por último la declaración de está API, que recolectará los datos de la base de datos hacia la página.

```
    const firebaseConfig = {

  apiKey: "AIzaSyCfWGfOvfmv7aV74H5UzTjeNZZjMbfDfTU",
  authDomain: "gps-tracker-710.firebaseapp.com",
  databaseURL: "https://gps-tracker-710-default-rtdb.firebaseio.com",
  projectId: "gps-tracker-710",
  storageBucket: "gps-tracker-710.appspot.com",
  messagingSenderId: "529606423358",
  appId: "1:529606423358:web:2148437f69544c74ebb26f",
  measurementId: "G-VQ7VYJVQ0W"

};

window.initialize = initialize;

firebase.initializeApp(firebaseConfig );

var ref = firebase.database().ref();

ref.on("value", function(snapshot) {
    var gps = snapshot.val();
  console.log(gps.LAT);
  console.log(gps.LNG);
  lat = gps.LAT;
  lng = gps.LNG;

  map.setCenter({lat:lat, lng:lng, alt:0});
  mark.setPosition({lat:lat, lng:lng, alt:0});

  lineCoords.push(new google.maps.LatLng(lat, lng));

  var lineCoordinatesPath = new google.maps.Polyline({
    path: lineCoords,
    geodesic: true,
    strokeColor: '#2E10FF'
  });

  lineCoordinatesPath.setMap(map);
}, function (error) {
  console.log("Error: " + error.code);
});
```

Figura 16. Parte del código de diseño de página web.

En la figura 16 se muestra la parte de código en donde se inicializa, de igual manera, la base de datos con sus credenciales respectivas de donde se va a importar los datos; se recopila la información de la base de datos, por medio, del comando snapshot.val() que se lo almacena como variable gps; por último para que esta variable se la convierta como variable interpretable por Google maps se aplica el comando console.log (), el cual al ser un comando de impresión se lo almacena en dos variables, lat y lng, que son las que se requiere para interactuar con la Api de Google maps.

4. **Diseño e implementación de centro de monitoreo remoto:**

Para el diseño de la figura 17 realizamos modificaciones en el diagrama de funcionamiento con respecto a la figura 10, debido a que la comunicación entre la

aplicación y la Raspberry es por red de internet, para la obtención de la información de los sensores de proximidad como la del módulo GPS.

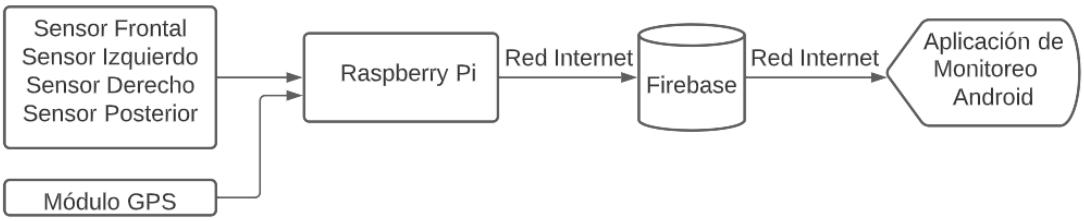

Figura 17. Diagrama de funcionamiento Aplicación de monitoreo Remoto.

A diferencia del centro de monitoreo local, para este caso se debe exportar tanto los datos recopilados por los sensores de proximidad como los datos de longitud y latitud obtenidos del módulo GPS.

El canal de comunicación que se utiliza para transportar información entre el centro de procesamiento hacia el centro de monitoreo es la red de internet; por medio de este medio, se envían los datos temporalmente hacia la base de datos en tiempo real, donde se almacenan momentáneamente mientras que la aplicación realiza la solicitud de importación de los datos de la base de datos hacia la pantalla que solicite el usuario. A continuación, se mostrará como se presenta la aplicación al usuario.

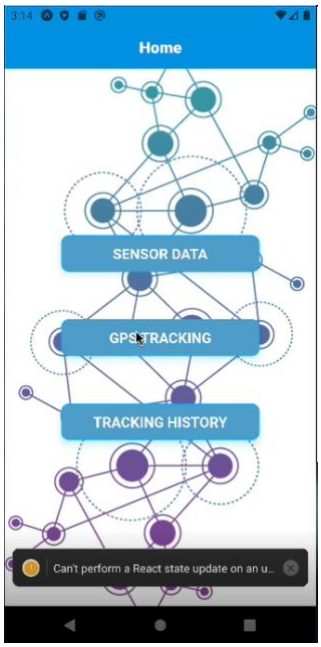

Figura 18. Pantalla Principal de Aplicación Android.

Esta aplicación cuenta con tres opciones, como se muestra en la figura 18, para el usuario: Información de los sensores de proximidad, rastreo GPS e historial de movimiento que realiza el vehículo. Mediante estas tres opciones se podrá tener un monitoreo simple pero útil, ya que se puede tener conocimiento de lo que pasa con el vehículo sin necesidad de encontrarnos en el mismo lugar, únicamente se debe tener en cuenta de que los dos dispositivos se encuentren conectados a la red de internet porque es el canal por el cual se transporta la información.

En la figura 19 se muestra la pantalla a la que dirige la aplicación para el monitoreo de los sensores de distancia. Esta figura muestra cómo se puede obtener datos de los cuatros sensores de proximidad y también, en la figura 19 se puede ver al igual que en la aplicación de monitoreo local en la aplicación móvil los cuadros y números de los sensores cambian de color cuando pasan los valores umbrales que se establecieron.

Figura 19. Pantalla de monitoreo para los sensores de proximidad.

Figura 20. Variación de colores de los sensores.

En la figura 20, se puede apreciar que los datos de los sensores frontal e izquierda se encuentran en rojo, debido a que se encuentran en valores menores a 100 centímetros que es el valor umbral en donde se le indica al usuario que está muy próximo a estrellarse; el posterior se cambia a color amarillo debido a que el obstáculo se encuentra a más de 100 centímetros pero a menos de 150 centímetros, lo cual quiere indicar que debe tener precaución porque se está acercando a un obstáculo y debe empezar a evitarlo; por último se encuentra el sensor derecho que se encuentra en color verde porque el valor es mayor a 150 centímetros lo cual indica que no existe un obstáculo que se deba esquivar.

En la figura 21 se muestra la prueba que se realizó con el vehículo autónomo, como se puede apreciar en la figura la persona que se encuentra en la parte frontal del vehículo es el obstáculo que se reconoce por el sensor frontal y si nos fijamos en la distancia, se puede notar que el brazo se encuentra a una distancia muy próxima al vehículo (de 10, 57 centímetros aproximadamente, como se muestra en la

aplicación), por lo tanto el que controla puede saber que se aproxima un obstáculo y debe curvar para evitarlo; por otro lado, tenemos el sensor del lado posterior que se encuentra en color amarillo debido a que la segunda persona se encuentra en la parte posterior y el vehículo capta mediante el sensor que hay un obstáculo atrás que está más lejos que el obstáculo frontal pero que se debe tener precaución al momento de esquivar, ya que si se dirige hacia atrás se acercaría al otro obstáculo y podría chocar.

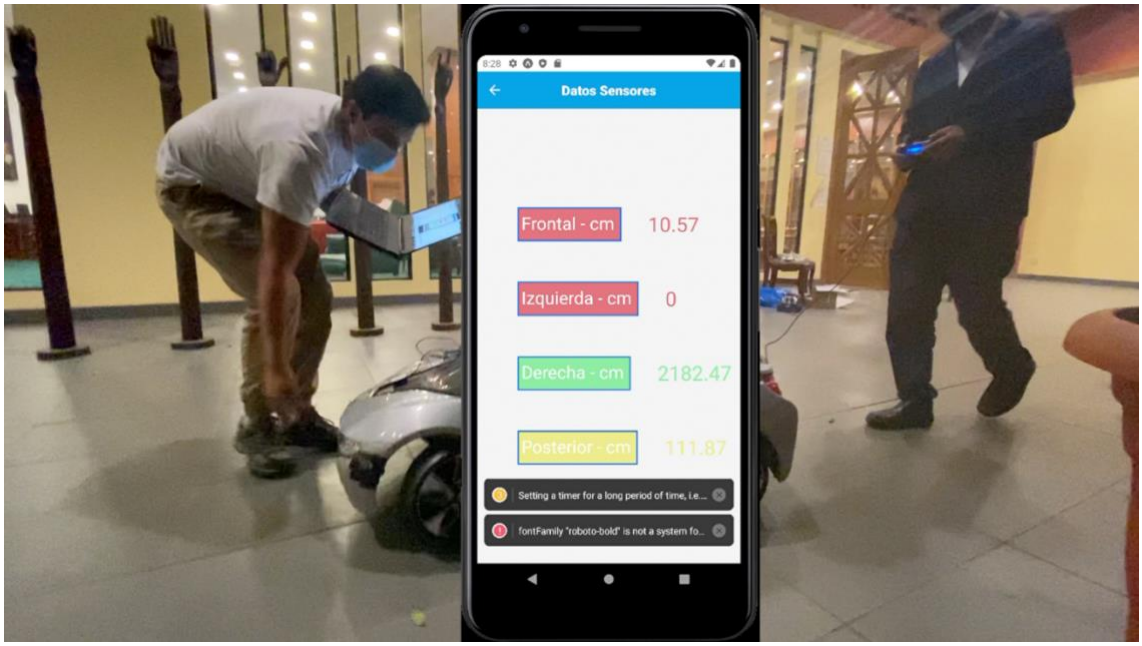

Figura 21. Prueba de funcionamiento del monitoreo de los sensores mediante la aplicación.

Retomando el funcionamiento de la aplicación, en la figura 22 se muestra el uso del rastreo del vehículo por medio de la aplicación, en donde se puede apreciar que el vehículo se traslada de un punto a otro.

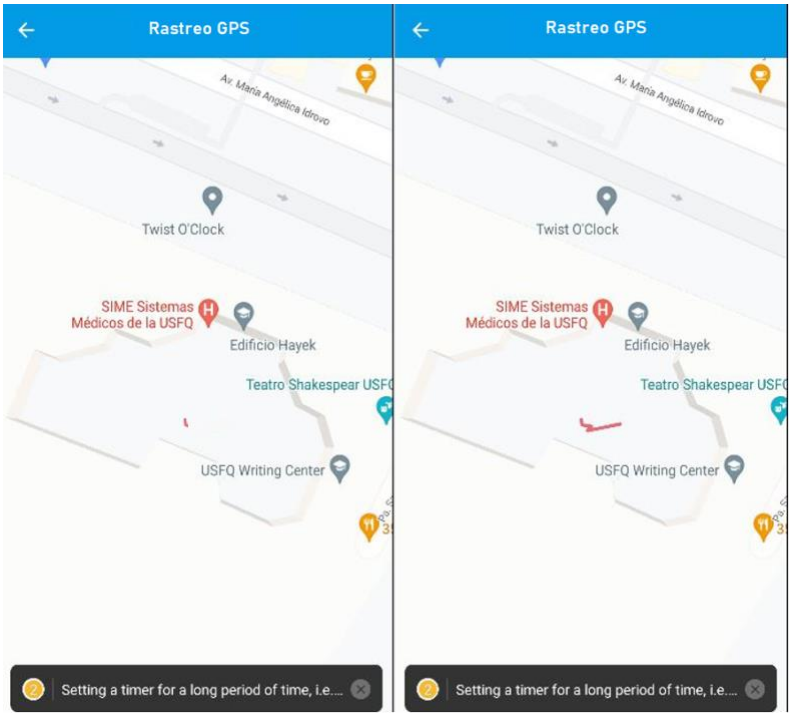

Figura 22. Rastreo GPS.

La trayectoria descrita en la figura 23, se almacena en la aplicación usando el almacenamiento del celular y de esta forma al momento de seleccionar la opción del historial del GPS se puede tener una lista con las rutas descritas por el vehículo, en la figura 23 se muestra como desde el historial del GPS se accede a la ruta que se realizó en la prueba que se implementó con el vehículo.

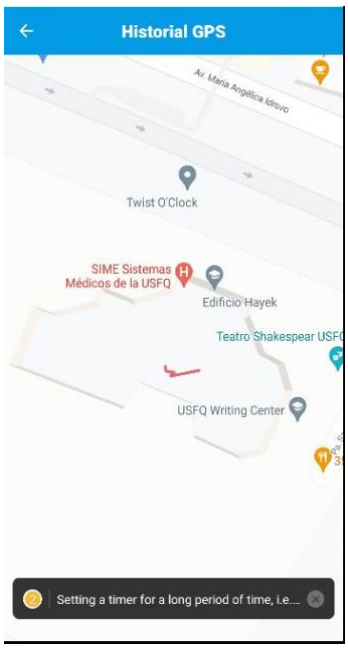

Figura 23. Historial Rastreo GPS realizado en la USFQ.

Esta aplicación de monitoreo se pudo realizar por medio de programación en Javascript y por medio del uso de visual code que fue el entorno en el cual se programó. En la figura 24 se muestra cómo se realiza la programación para la aplicación.

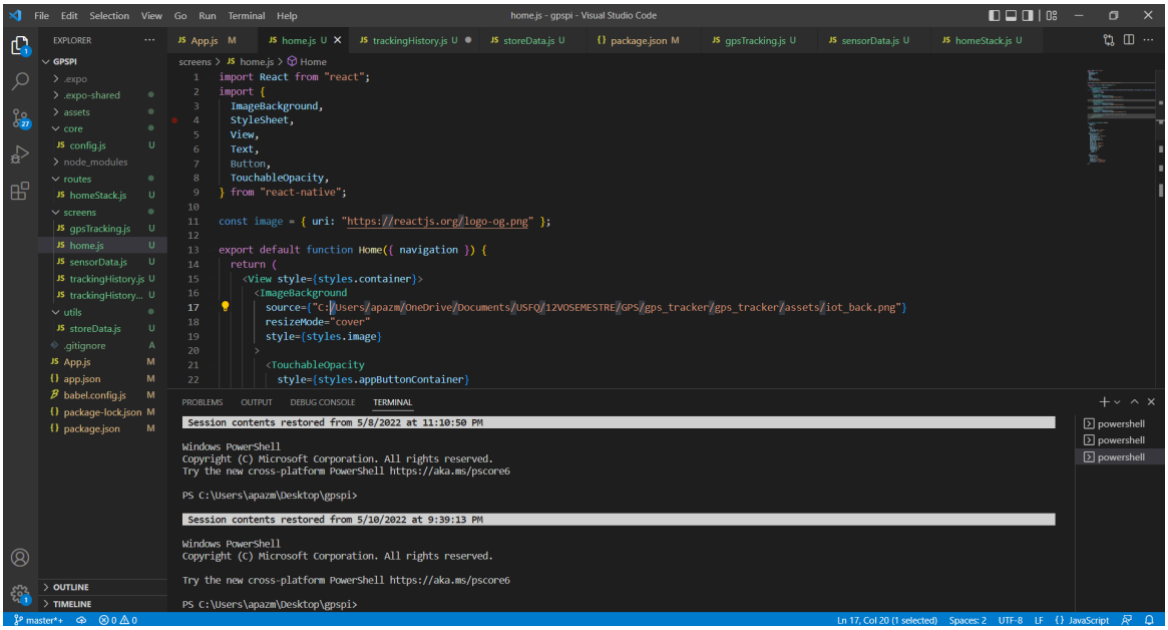

Figura 24. Visual Code.

Para la transportación de los datos se utiliza internet como canal de comunicación y la base de datos que se muestra en la figura 25 es el fichero que permite almacenar la información para luego por medio de internet transferirlo a la aplicación móvil.

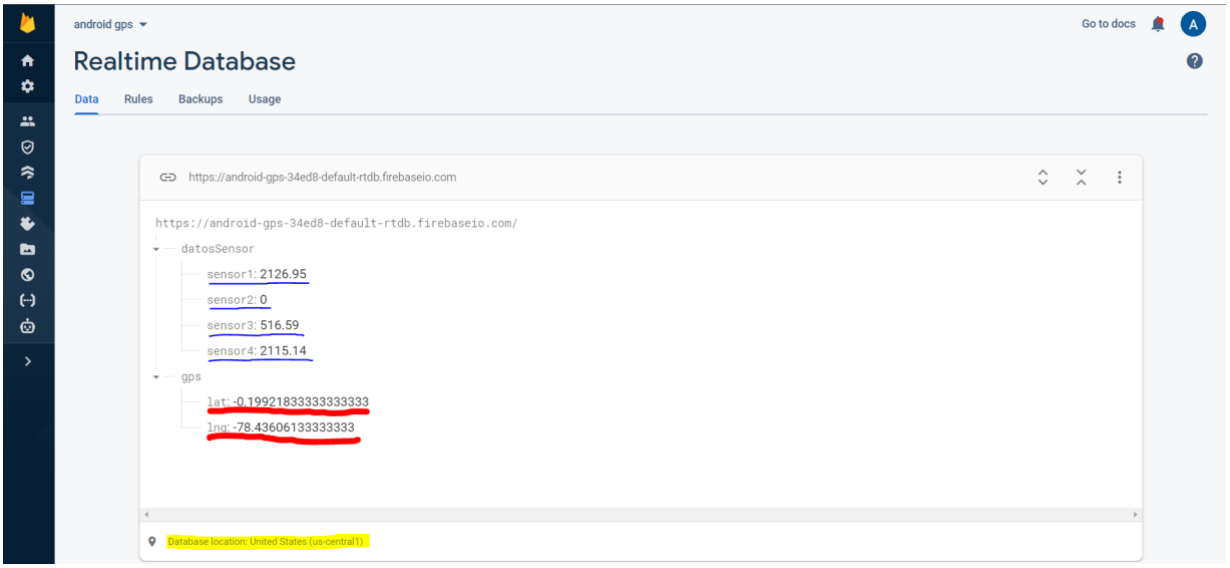

Figura 25. Base de datos de Google.

La aplicación móvil tiene como elementos importantes las variables que se van a obtener de los sensores y las variables de longitud y latitud que se obtiene del módulo GPS, estas variables son las que se utilizarán en la base de datos; en la cual se enviarán datos desde la Raspberry pi, que es el centro de procesamiento del vehículo, hacia la base de datos; en la cual, como se aprecia en la figura 25, con color azul se encuentran las variables de los sensores de proximidad, con color rojo las variables del módulo GPS y con color amarillo el servidor que se utiliza para almacenar y transferir la información. El servidor está ubicado en Estados Unidos, lo cual será un limitante al momento de tener velocidad en el tiempo de respuesta de la aplicación, debido a que la latencia de transportación de datos es de aproximadamente 100 ms, resultado obtenido mediante la realización de prueba de ping que se observa en la figura 26. Por lo tanto, debido a este problema la aplicación por naturaleza va a tener retardo, lo cual afectará a la eficacia de la obtención de datos instantáneos.

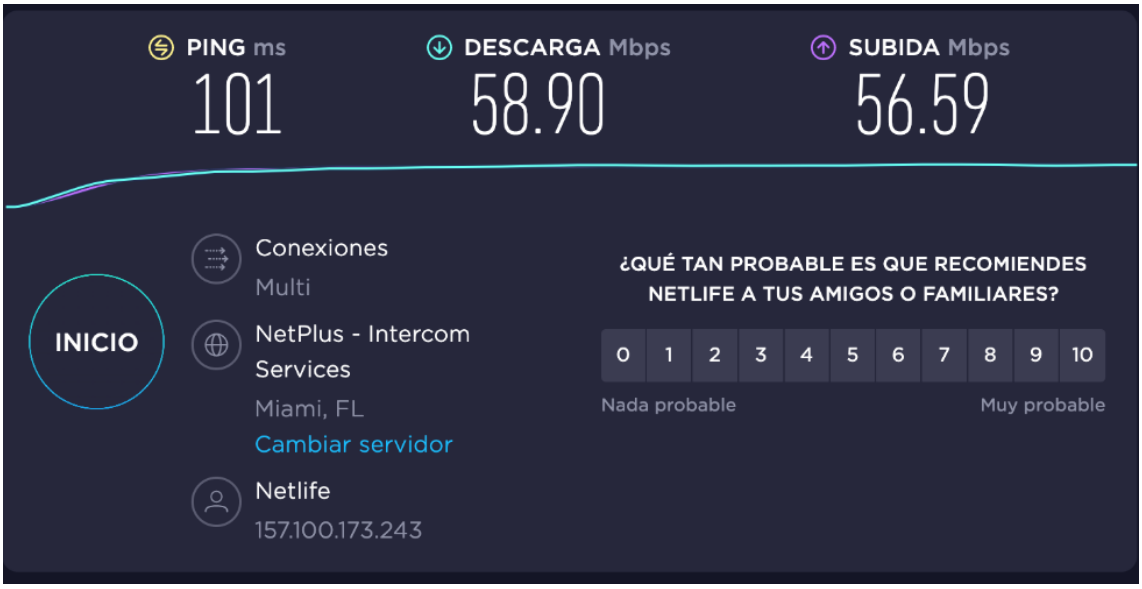

Figura 26. Prueba de velocidad de transportación de datos, mediante página web Ookla.

Finalmente, para poder reconocer otros tipos de retardos en la aplicación se desarrollaron pruebas de latencia locales de uno de los sensores que se implementó en el carrito para sumarle a la latencia natural y determinar un valor muy próximo de la latencia que se tendrá en el monitoreo remoto, con la finalidad de trabajar a futuro en la reducción de estos retardos.

Figura 27. Prueba de latencia sensor de proximidad HCSR04.

En la figura 27 se muestran los primeros 37 datos obtenidos de la prueba de retardo que se realizó en el sensor de proximidad, para esta prueba se obtuvieron 29 datos que arrojaron un retardo de aproximadamente 3 milisegundos, 4 de 30 milisegundos y 1 de 180 milisegundos. Los datos de color verde son datos que no son imperceptibles ni afectan en el funcionamiento, pero los datos amarillos y los datos rojos afectan significativamente al sistema debido a que estos problemas llevan al programa desarrollado en la Raspberry que controla el vehículo, que entre en condiciones de carrera, con un 78% de datos eficientes y un 22% de datos con mayor retardo de lo esperado, lo cual es significativo y determinante al momento de ejecutar el programa. Una condición de carrera es un comportamiento del software en el cual la salida depende de un orden de ejecución de eventos que no se

encuentran bajo control y esto provoca resultados incorrectos en el programa. (Román, 2019)

Por lo tanto, la aplicación de monitoreo remoto tendrá una variación de latencia de entre 103 milisegundos hasta 280 milisegundos, los cuales son tiempos perceptibles y que si bien para el monitoreo no afecta para el momento de pasar a una aplicación de control remota son tiempos que se deben tomar en cuenta y se deben tratar de reducir lo mayor posible.

CONCLUSIONES

- En la actualidad el internet de las cosas o IoT, es la nueva revolución industrial que se tiene y por ende esta aplicación lo que busca es involucrarse y tener un acercamiento a lo que va a ser al futuro, ya que por medio de esta aplicación se puede obtener información del vehículo sin necesidad de encontrarnos ubicados en el mismo lugar. De este modo podríamos incluir al vehículo entre nuestros dispositivos conectados a nuestro celular de los cuales se tiene monitoreo.
- Un problema muy importante al momento de diseñar e implementar la aplicación móvil es el problema de retardo que presentan cada uno de los sensores ya que estos retardos ocasionan que el programa de procesamiento del vehículo entre en condiciones de carrera y envíe datos erróneos a la aplicación de monitoreo remoto, ya que con información incorrecta la persona que está monitoreando y manejando el dato no podrá tener confianza de los resultados que observe en la pantalla y por ende no podrá maniobrar conforme los datos son transmitidos.
- El sistema de georreferencia y localización del vehículo es bueno ya que se puede obtener un rastreo no solo de ubicación sino también de movimiento del vehículo, lo cual es bueno ya que ahora se cuenta con un módulo más en el vehículo que puede servir para desarrollar otras aplicaciones en el mismo.
- Durante el desarrollo de las aplicaciones de monitoreo se pudo reconocer que para usar el módulo GPS se debe establecer una comunicación serial, al igual que el bluetooth del mando de ps4 que se utiliza para manejar el vehículo; por lo que se pudo determinar que la Raspberry únicamente cuenta con un

solo puerto de comunicación serial. De este modo, el módulo GPS transmite señales digitales de transmisión y recepción.

- Luego de realizar pruebas independientes de los sensores de proximidad se llega a la conclusión de que, si bien los sensores son muy precisos, en ciertos momentos presentan retardos, de hecho el 22% de los datos obtenidos en la muestra son datos que pueden ocasionar condición de carrera en el programa de procesamiento, lo cual en término de programación es la peor condición en la que se puede encontrar el programa ya que los resultados que envía a partir de ahí vienen a ser incorrectos hasta que el programa salga de este periodo de carrera que es indeterminado.

RECOMENDACIONES

- Para próximos proyectos se deben tomar en cuenta los tiempos de retardo de cada sensor que se vaya a implementar y adaptar la programación al tiempo de retardo del sensor más lento del sistema ya que el sensor más lento podría ocasionar problemas en el programa de procesamiento y control del vehículo.
- Para próximos proyectos se pueden desarrollar otro tipo de prácticas con el módulo GPS para intentar conseguir que el vehículo se pueda transportar de un punto A a un punto final B de forma autónoma, ya que por el momento únicamente se tiene un sistema de evasión de obstáculos autónoma.

REFERENCIAS

Educación 3.0. (27 de Diciembre de 2018). *EDUCACIÓN 3.0* . Obtenido de Raspberry Pi, el ordenador perfecto para educación: https://www.educaciontrespuntocero.com/noticias/raspberry-pi-educacion/

Clavijo Rujel, J., 2022. *Geolocalizador portátil con Raspberry Pi 3 y procesamiento de datos en QGIS*. [online] Repositorio.unp.edu.pe. Disponible en: <https://repositorio.unp.edu.pe/handle/20.500.12676/2687> [Accedido el 5 abril 2022].

CONDORI CALIZAYA, I., 2022. *"APLICACIÓN DE LOCALIZACIÓN Y CONTROL DE DISPOSITIVOS SMARTPHONE ANDROID"*. [online] Repositorio.umsa.bo. Disponible en: <https://repositorio.umsa.bo/bitstream/handle/123456789/7936/T.2827.pdf?sequence=1&isAllowed=y> [Accedido el 7 abril 2022].

Criollo Merino, F. (2018). *IMPLEMENTAR UN SISTEMA PARA LA DETECCIÓN DE PRESENCIA EXTERNA PARA VEHÍCULOS PESADOS PARA PREVENCIÓN DE ACCIDENTES*. Repositorio.unp.edu.pe. Recuperado el 7 abril de 2022, en https://repositorio.unp.edu.pe/bitstream/handle/UNP/1306/CIE-CRI-MER-17.pdf?sequence=1&isAllowed=y.

Sanner, M. F. (e.g. 1975). PYTHON: A PROGRAMMING LANGUAGE FOR SOFTWARE INTEGRATION AND DEVELOPMENT. The Scripps Research Institute. e.g. 32 (e.g. 2), pp.7.

Eguíluz, J. (2009). Introducción a Javascript. LibrosWeb. Obtenido de https://www.jesusda.com/docs/ebooks/introduccion_javascript.pdf

Román Díez, G. (2019). *Concurrencia Condiciones de Carrera*. Babel.upm.es. Retrieved 21 April 2022, from http://babel.upm.es/teaching/concurrencia/material/slides/groman/CC_CondCarrera.pdf.

Raspberry Pi 4 Model B. (21 de J de 2019). Obtenido de https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

Hernandez, C., Barbosa, J., Paredes, D., & Játiva, R. (2 de Diciembre de 2020). Design and Implementation of an autonomous vehicle with LIDAR-based navigation. Obtenido de IEEE Xplore: https://ieeexplore.ieee.org/document/9359423