UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingeniería

Implementación de prototipo de Centro de Monitoreo Remoto de un Vehículo Terrestre No Tripulado.

Andrés Gustavo Barba Calderón

Ingeniería Electrónica

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

Quito, 24 de noviembre de 2021

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingeniería

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Implementación de prototipo de Centro de Monitoreo Remoto de un Vehículo Terrestre No Tripulado

Andrés Gustavo Barba Calderón

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

Quito, 24 de noviembre de 2021

© DERECHOS DE AUTOR

Nombres y apellidos: Andrés Gustavo Barba Calderón

Código: 00125373

Cédula de identidad: 1718866732

Lugar y fecha: Quito, 24 de noviembre de 2021

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UNPUBLISHED DOCUMENT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 dirige a la implementación de un prototipo de centro de monitoreo remoto de un vehículo terrestre no tripulado. Particularmente, este trabajo complementa otro relacionado con un prototipo de vehículo con capacidades de navegación autónoma. La interfaz gráfica desarrollada en este proyecto permite visualizar remotamente el trabajo de los sensores dispuestos en un vehículo no tripulado mediante el uso de sockets que vinculan al dispositivo de monitoreo y al CPU del vehículo dentro de una red inalámbrica. Se extrae información de sensores de proximidad, gps y de dos cámaras instaladas en el vehículo. Se visualiza: condiciones de obstrucción a partir de los sensores de proximidad, geolocalización del vehículo en términos de latitud y longitud y vistas del entorno obtenidas de las cámaras frontal y trasera del carrito. Todo este proceso se efectuó en un microcomputador Raspberry Pi programado en Python.

Palabras claves: Vehículo autónomo, GPS, cámaras, Sensores de proximidad, Python, Sockets.

ABSTRACT

This project is aimed to the implementation of a prototype of a remote monitoring center for an unmanned land vehicle. In particular, this work complements another related to a vehicle prototype with autonomous navigation capabilities. The graphical interface developed as part of this project allows to remotely view the work of the sensors available in an unmanned vehicle through the use of sockets that link the monitoring device and the vehicle's CPU within a wireless network. Information is extracted from proximity sensors, gps and two cameras installed in the vehicle. Obstruction conditions from the proximity sensors, geolocation of the vehicle in terms of latitude and longitude and views of the environment obtained from the front and rear cameras of the cart are provides. This whole process was carried out on a Raspberry Pi microcomputer programmed in Python.

Keywords: Autonomous vehicle, GPS, cameras, Proximity sensors, Python, Socket.

TABLA DE CONTENIDO

Contenido

ÍNDICE DE FIGURAS

Índice Tablas

INTRODUCCIÓN

El presente trabajo trata sobre la creación de un centro de monitoreo remoto, su función principal es la recolección de datos de los sensores instalados en un vehículo terrestre no tripulado, sin afectar el ejercicio de sus funciones, además de añadir geo-referenciación al mismo. La conectividad que subyace en estos dos proyectos recae en la recepción directa que hace el centro de monitoreo del estado de cuatro sensores que se encuentran conectados al vehículo autónomo. Estos sensores de proximidad receptan información de posibles obstáculos y advierten de posibles amenazas durante la navegación. Por otra parte, el dispositivo de monitoreo incluye un módulo GPS que determina la latitud y longitud correspondientes a la posición del vehículo que se visualiza en la interfaz gráfica del centro de monitoreo, conjuntamente con imágenes del entorno del carrito proporcionadas por cámaras ubicadas al frente y en la parte posterior del mismo.

La inspiración y relevancia del presente proyecto forma sus cimientos en la necesidad de estructurar una mayor seguridad para los conductores y una mayor seguridad a nivel común en esta esfera vehicular de contacto social. Es preciso tomar en cuenta que el actuar humano contempla la posibilidad de fallo o error, puesto que el mismo desde su estructura personal no posee perfección, es, por tanto, óptimo propender a una cultura vehicular apoyada en la tecnología actual que responsablemente y en forma progresiva doten de plena seguridad vehicular a nuestra sociedad. En un futuro, la esfera tecnológica será la encargada de regular la conducción, y de eliminar los posibles peligros para los usuarios del sistema de transportación. Este proyecto nos aproxima a la posible realidad de una sociedad enteramente automatizada.

En la actualidad la conducción autónoma ha alcanzado altos niveles de evolución y se continúa trabajando en su perfeccionamiento. Los aportes de este transporte del futuro son la seguridad, comodidad y culturización tecnológica. El día que los coches autónomos logren conquistar las carreteras, va ser notable la reducción de los accidentes, hasta casi su extinción. Esto se debe a que la tecnología va a ser testada y no se podrán quebrantar las normas básicas de conducción, además de proveer medios de comunicación cooperativa entre los vehículos del entorno, reduciendo la probabilidad de adelantamientos peligrosos, etc.

En un sistema de navegación, es muy importante la observación del entorno cuando un vehículo está en movimiento, ya sea de forma manual o automática. En todos los casos se busca una clara orientación de lo que sucede delante del auto y en la parte posterior. En este proyecto, se utiliza programación en Python para desarrollar un código que, con el uso de cámaras, permite la orientación espacial del vehículo. El espectador remoto pueda observar que está pasando tanto en la parte frontal como posterior, y podría evitar cualquier percance o accidente accionando el modo de navegación por control remoto disponible en el auto.

Marco Teórico

Sensor de Proximidad HC SR04

Se muestra en la Figura 1, es mejor conocido como sensor ultrasónico HC SR04. Su función es ayudar a la medición de distancias mediante sus transductores que son dos: un altavoz y un micrófono. Esto va a generar pulsos que van a ser de alta frecuencia que va a rebotar en objetos que se encuentren cerca y se va a reflejar hacia el sensor que va a ser captado por medio del micrófono. Son sensores que son muy útiles para los universitarios porque son económicos y amigables con el usuario.

Con frecuencia se usan en el campo de la robótica para poder detectar obstáculos, para poder determinar posiciones, crear mapas entre otros.

Figura 1: Sensor de Proximidad HC SR04

Webcams

Se muestran en la Figura 1. Una webcam es una cámara digital pequeña que se conecta a una computadora en este caso al Raspberry pi y nos ayuda a capturar imágenes o videos y trasmitirlas a la web o a la computadora misma de una manera privada.

En este caso como se dijo anteriormente se conecta a la Raspberry mediante un puerto USB. Debidamente sincronizada y conociendo en la ventana de control cual interfaz se está utilizando, nos permite observar el entorno de navegación con una buena calidad de imagen.

Figura 2: Webcam

Para el reconocimiento de la webcam en el Raspberry se usa los siguientes comandos. Sudo apt-get install fswebcam y Fswebcam image.jpg, después de haber configurado en la raspi config su actividad.

Dualshock 4 wireless controller

Es un control inalámbrico de la consola de Sony Play Station 4 que en este proyecto se utilizó para controlar el auto de forma manual. Se conecta mediante bluetooth al procesador del Raspberry pi y a partir de comandos previamente establecidos se controla a voluntad el desplazamiento del carrito.

Python:

Es un lenguaje de programación de alto nivel usado en el desarrollo de todo tipo de aplicaciones. Se lo puede definir como un lenguaje interpretado. En otras palabras, no es necesario compilar para que el programa pueda ejecutarse, no es necesario que se lo traduzca a lenguaje de máquina porque utiliza un programa llamado interpretador, Cabe recalcar que es un lenguaje multiplataforma y también de código abierto.

Figura 3: Logo Python

Sockets Python:

Los Sockets Python, proveen una interfaz de comunicación. Es una tecnología de programación mediante la cual se pretende conectar dos nodos a una red para lograr una comunicación entre sí. La primera conexión se escucha en un puerto particular mientras que el otro se acerca para formar una conexión. A manera de ejemplo, se podría decir que es la columna vertebral de la navegación web mediante la cual se interrelaciona un servidor y un cliente.

Centro de Monitoreo:

Los centros de monitoreo se conocen como espacios de interacción entre operadoras mismas que mediante la presentación de datos o Videos simultáneamente expuestos permiten al usuario captar la atención de diversas esferas en un solo punto. Se podría definir, por tanto, a los centros de monitoreo como lugares de convergencia para la información. Un claro ejemplo son los sistemas de seguridad que facilitan cobertura preventiva.

Su principal función es la recepción, el procesamiento y la guarda de toda la información codificada de los mecanismos de seguridad. Al mantener la visión clara y completa de todos los elementos que intervienen en los procesos operativos, permite al usuario tomar decisiones informadas en el momento adecuado. Por tanto, un centro de

operaciones de monitoreo posee una correcta visión de las instalaciones, los protocolos de organización, coordinación y control eficiente.

Módulo GPS:

Se trata de un dispositivo electrónico que utiliza las señales enviadas por satélites para determinar con exactitud, su posición sobre la superficie terrestre. El módulo receptor opera en la banda de comunicación satelital, utiliza micro controladores y emplea técnicas de trilateración que en el mismo proceso permiten determinar ubicación velocidad y elevación.

Interfaz Gráfica:

La interfaz gráfica consiste en un programa de carácter informático que actúa como interfaz de usuario. Se define como un programa tecnológico estructurado por un conjunto de objetos gráficos y muchas imágenes que representan la información y las acciones disponibles en la mencionada interfaz. La comunicación que se da entre los dos dispositivos tecnológicos permite la proyección de uno a otro con el fin de canalizar la información hacia el usuario. Su principal uso consiste en proveer un entorno visual para permitir la comunicación óptima con un sistema operativo del aparato tecnológico transmisor.

Este software de carácter visual evidencia lo que conlleva todas las posibles acciones en la plataforma destino, así como también la información disponible para que el usuario pueda comprender cuestiones informáticas sin necesidad de un estudio en el campo. En la actualidad esta interfaz gráfica se encuentra visible en absolutamente todas las páginas web con las que interactuamos de manera directa. Su principal finalidad es proporcionar comodidad al usuario y simplificar su interacción frente al dispositivo informático.

Desarrollo del Proyecto

1.1 Protocolo 802.11 n

El protocolo 802.11 es un estándar internacional que es definida por la IEEE (Institute of Electrical and Electronics Engineers) y hacen referencia a las tecnologías inalámbricas, los mismos que determinan los parámetros sobre la interfaz en el aire entre dos clientes o una estación. La ventaja principal radica en que no se necesita licenciamiento con respecto al respecto de frecuencia y son de libre uso.

Con respecto a el protocolo 802.11 n, es un estándar que tiene una tasa máxima de transferencia en 2.4GHz de 100Mbps superando a 802.11b y 802.11g. Este estándar es compatible con los dispositivos móviles y con la Raspberry Pi en lo que compete el presente proyecto.

1.2 NMAP

Nmap es una utilidad gratuita y de código abierto para el descubrimiento de redes. Utiliza paquetes IP sin procesar para determinar qué hosts están disponibles en la red, qué servicios ofrecen esos hosts, qué sistemas operativos están ejecutando, qué tipo de filtros de paquetes/firewalls están en uso, y docenas de otras características.

1.3 RDP

Las siglas RDP responden a Remote Desktop Protocol. El protocolo RDP, entonces, permite que el escritorio de un equipo informático sea controlado a distancia por un usuario remoto Este protocolo es aplicable tanto en sistemas operativos Windows y Linux mediante varios softwares que permiten realizar una conexión remota, en el caso de este proyecto se utilizó VNC Viewer para dicho fin ya que se puede usar de forma offline es decir dentro de una red LAN o WLAN, así como también de forma online con conexión desde internet.

2. Centro de Monitoreo

Se visualiza su completo funcionamiento mediante el siguiente diagrama de Bloques.

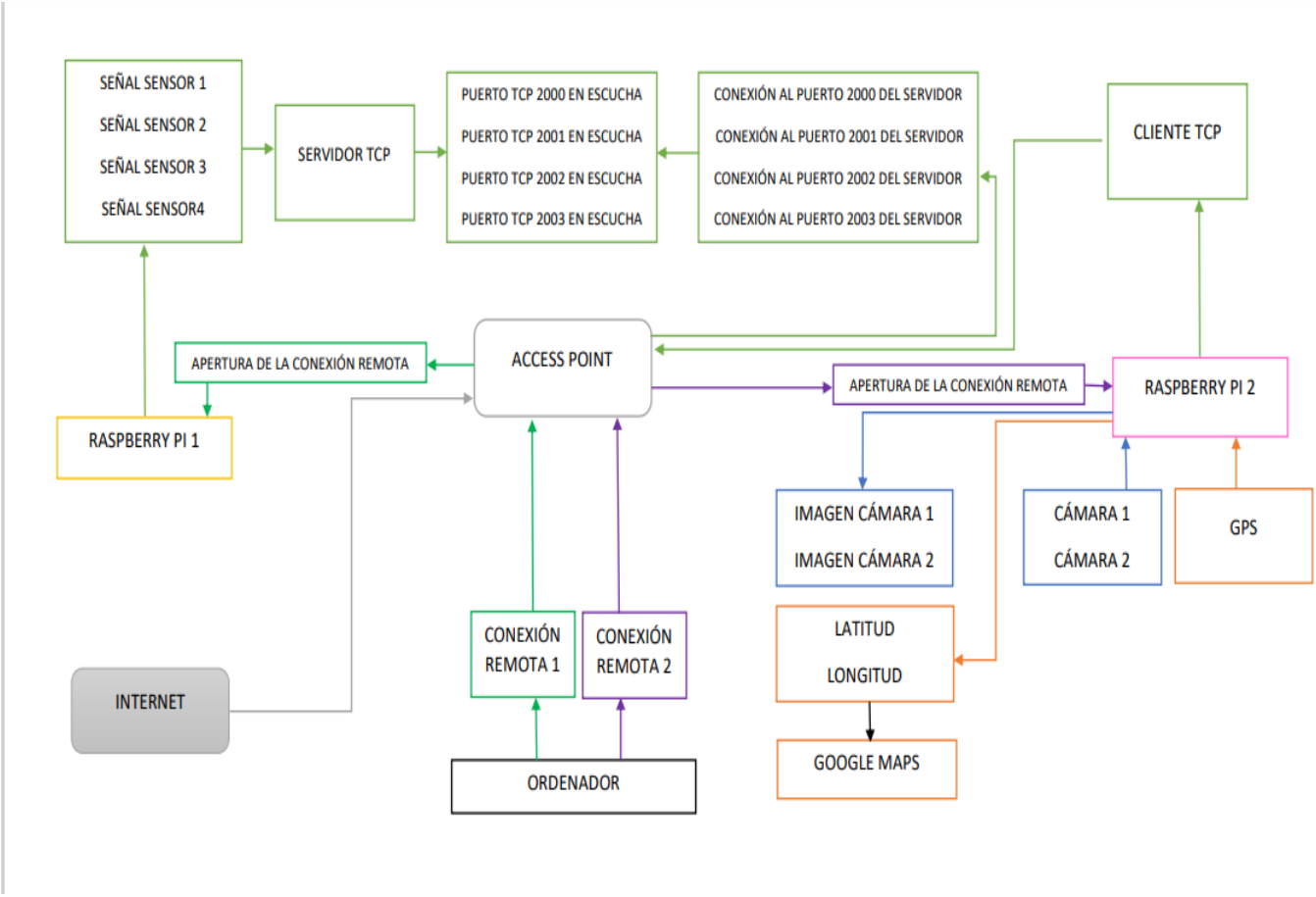

Figura 4. Diagrama de Bloques Centro de Monitoreo

2.1 Red WLAN

Para la visualización en tiempo real de las cámaras, sensores y geo posicionamiento se diseñó una red LAN inalámbrica en la cual los dispositivos ya descritos tuvieron que conectarse por medio del protocolo 802.11 n.

2.1.1 Conexión Offline

En primera instancia se tuvo que crear una red WLAN por medio de un Access Point sin acceso a internet como prueba de concepto. Para ello se utilizó el Access Point TP-LINK TL-WR841N en el cuál se definieron los siguientes parámetros de seguridad:

SSID	PASSWORD
BB10_USFQ_WF	123456789

Tabla 1. Parámetros de seguridad del Access Point.

Una vez definidos los parámetros de seguridad, se procedió a conectar la primera Raspberry Pi y la segunda Raspberry Pi respectivamente y un ordenador a la red del Access Point. Como se muestra en la siguiente figura

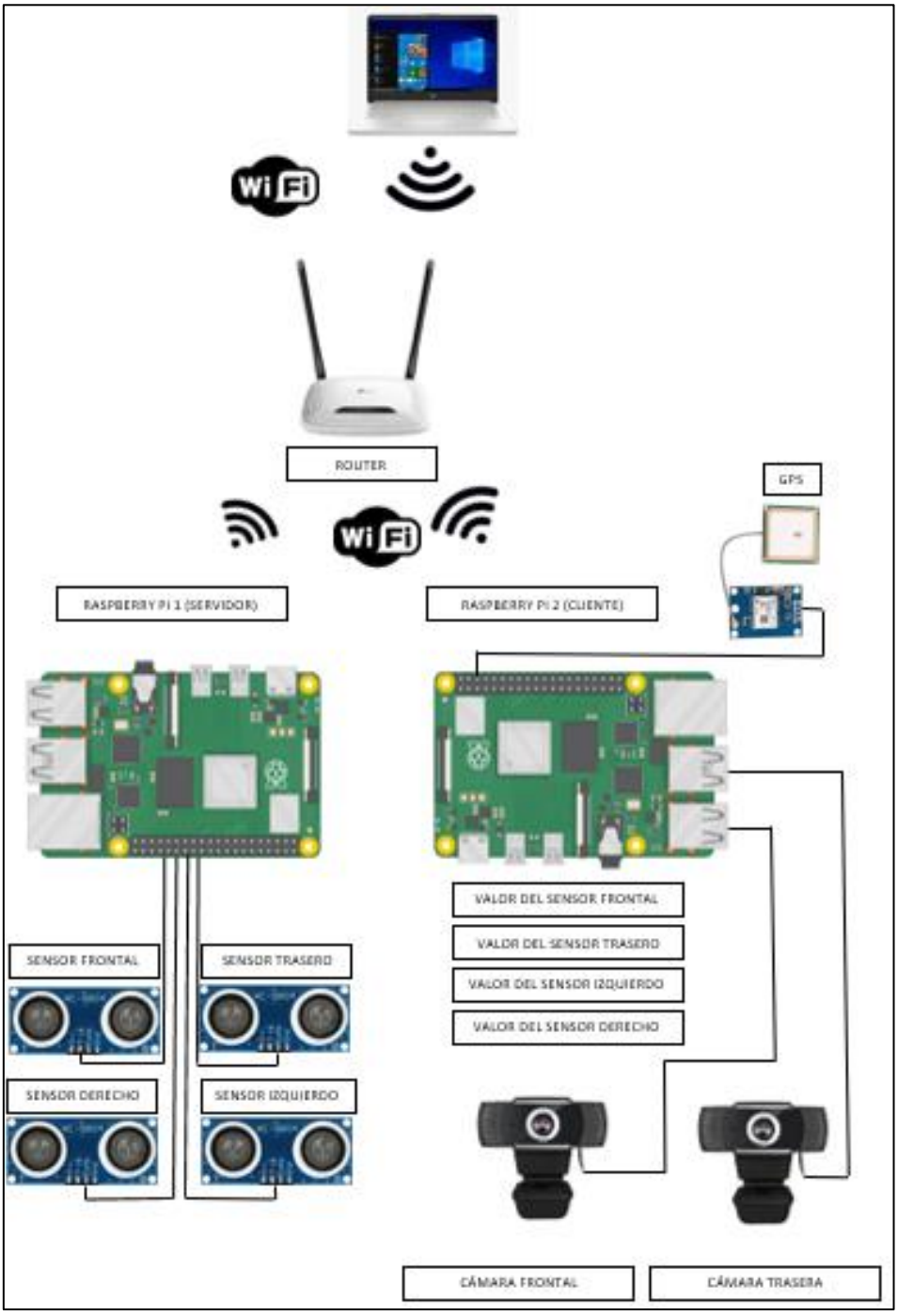

Figura 5. Conexión Offline.

Esta prueba de concepto se lo realizó con el fin de verificar la conexión de las dos Raspberry por medio de un acceso remoto.

2.1.2 Conexión Online

Una vez verificada la conexión Offline, se procedió a cambiar de Access Point para establecer el acceso a Internet, en este caso fue con el celular Xiaomi Redmi Note 8 ya que poseía datos móviles. Tal como en el caso del Acceso Point TP-LINK, se establecieron parámetros de seguridad descritos a continuación:

SSID	PASSWORD
Redmi Note 8-+	Password.Redmi

Tabla 2. Parámetros de seguridad del Access Point - Celular

La conexión con acceso a internet fue importante para abrir el link de Google Maps con los datos de latitud y longitud establecidas por el GPS desde la segunda Raspberry Pi como se muestra en la siguiente figura.

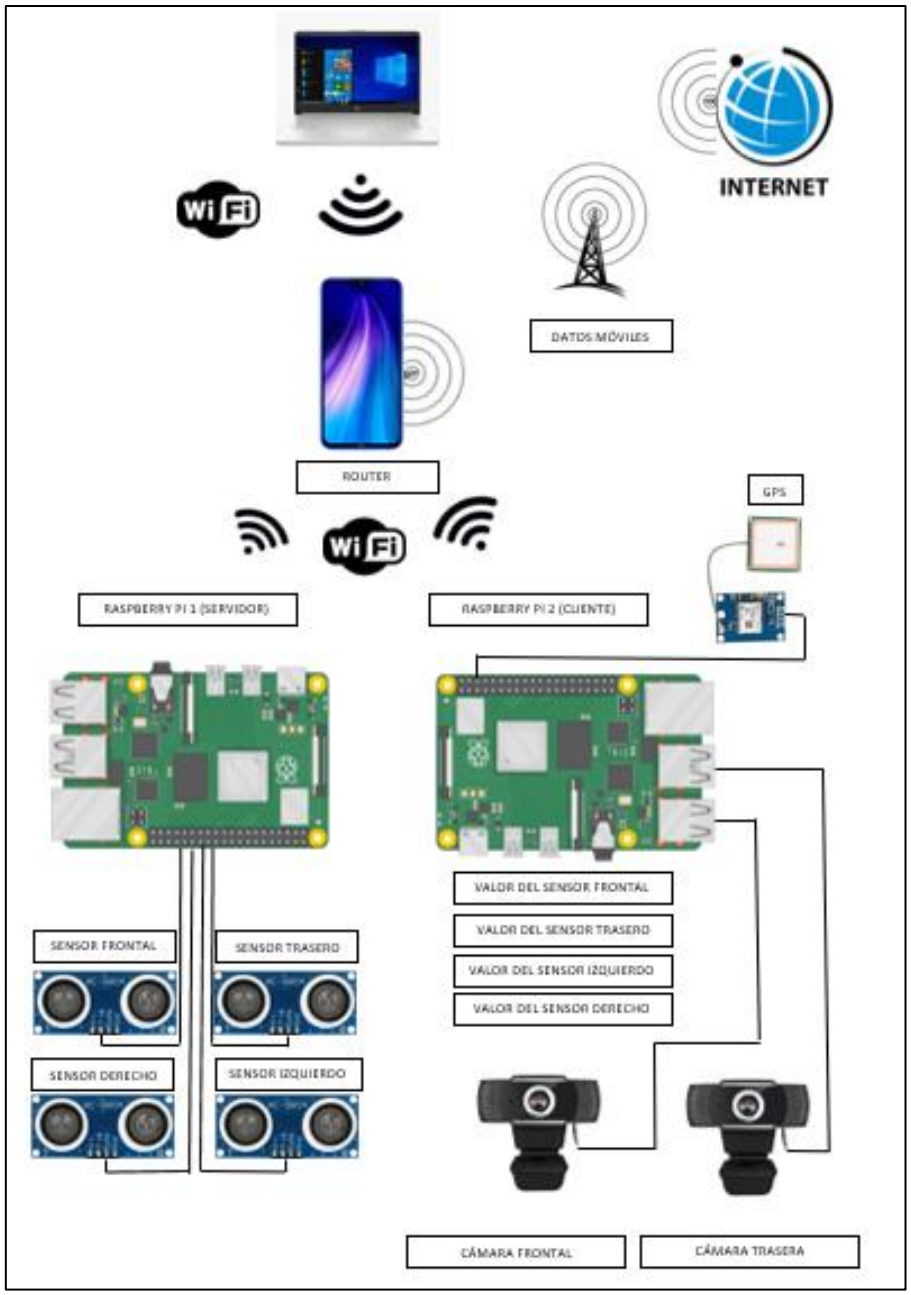

Figura 6. Conexión Online

2.2 Acceso Remoto

El acceso hacia la primera Raspberry Pi y la segunda Raspberry Pi se realizó mediante una conexión RDP desde un ordenador utilizando el software VNC Viewer. En primer lugar, se tuvieron que determinar cuáles eran las direcciones IP de cada Raspberry; para ello se usó la herramienta NMAP. El escaneo de hosts se lo realizó desde el ordenador que estuvo dentro de la misma red. Este proceso se lo realizó tanto en la

conexión offline (Figura 5) como en la conexión online (Figura 6) ya que las direcciones IP que proporcionan los Access Point no están en los mismos segmentos de red.

Una vez verificadas las direcciones IP de cada Raspberry, se procedió a colocar las direcciones IP, el usuario y contraseña para finalmente establecer la conexión. El software VNC Viewer por defecto utiliza el puerto 5900 TCP para la conexión RDP en cada Raspberry Pi.

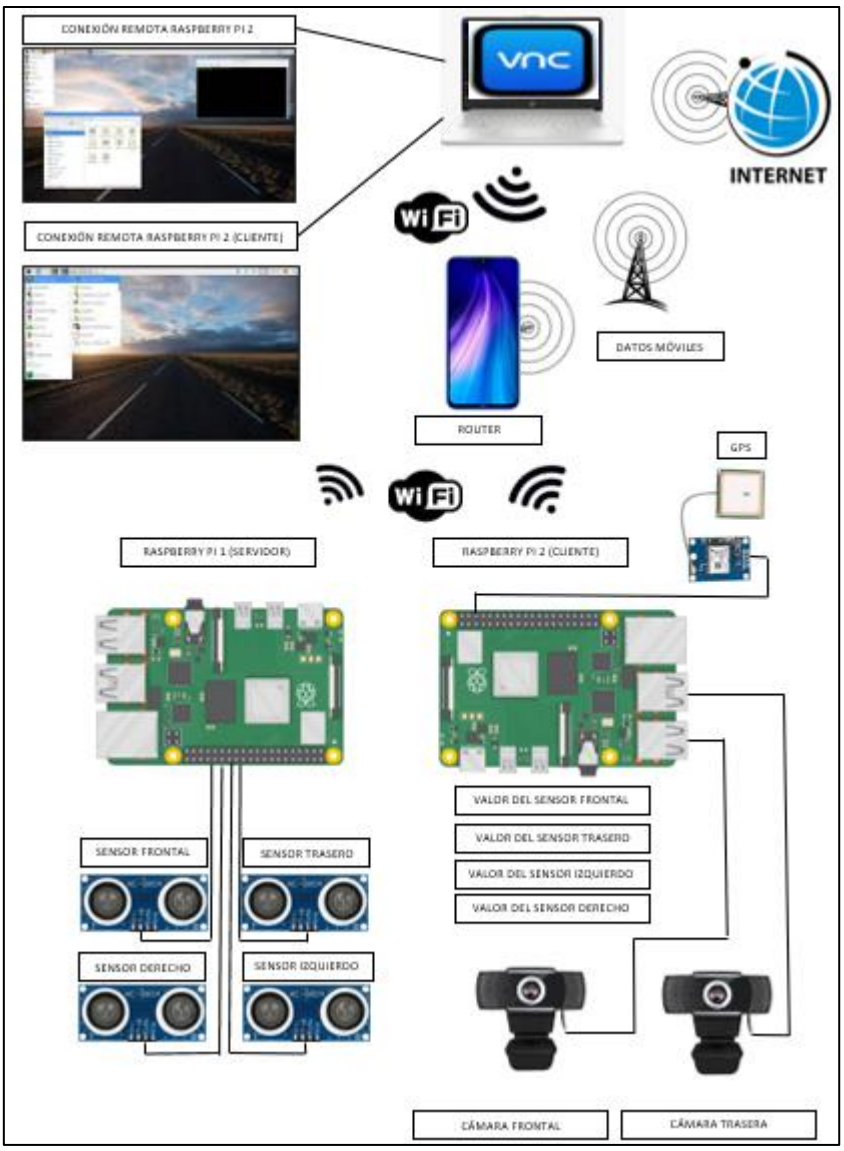

Figura 7. Acceso Remoto Online.

Se tiene la conexión exacta del centro de monitoreo con todos sus componentes sensores, GPS y cámaras, se muestra la conexión VNC que se realiza por medio de la dirección IP a los dos Raspberrys Pi

3. Servidor TCP/IP

En el caso de las conexiones físicas que existen entre los sensores ultrasónicos y la primera Raspberry Pi son usadas para procesar las señales y mostrarlas en una ventana interactiva. Estas conexiones físicas no se las puede realizar en paralelo con la segunda Raspberry Pi utilizada en el presente proyecto ya que disminuiría la intensidad de señal y actuaría como un divisor de voltaje, generando fallos de lectura. Para resolver este inconveniente, se montó un servidor TCP/IP utilizando la librería socket de Python en la primera Raspberry Pi con 4 puertos en escucha para que realice el envío de las señales procesadas hacia un cliente TCP/IP que en este caso es la segunda Raspberry Pi. Para este fin, se estableció una red LAN mediante un router para la interconexión del Servidor-Cliente (Figura 8).

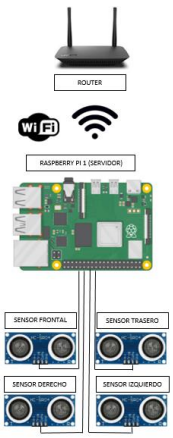

Figura 8. Servidor TCP/IP

El código se puede ver en la sección de ANEXOS Código fuente para el servidor TCP/IP al igual que su respectivo diagrama de bloques.

4. Cliente TCP/IP

En el caso del cliente TCP/IP, también se utilizó la librería socket de Python para establecer la conexión remota hacia el servidor y recibir, en primera

instancia, los bytes desde el servidor. Luego mediante la decodificación a caracteres Unicode utf-8, se visualizaron los datos de lectura de los sensores ultrasónicos a través de sockets de conexión TCP/IP (Figura 9).

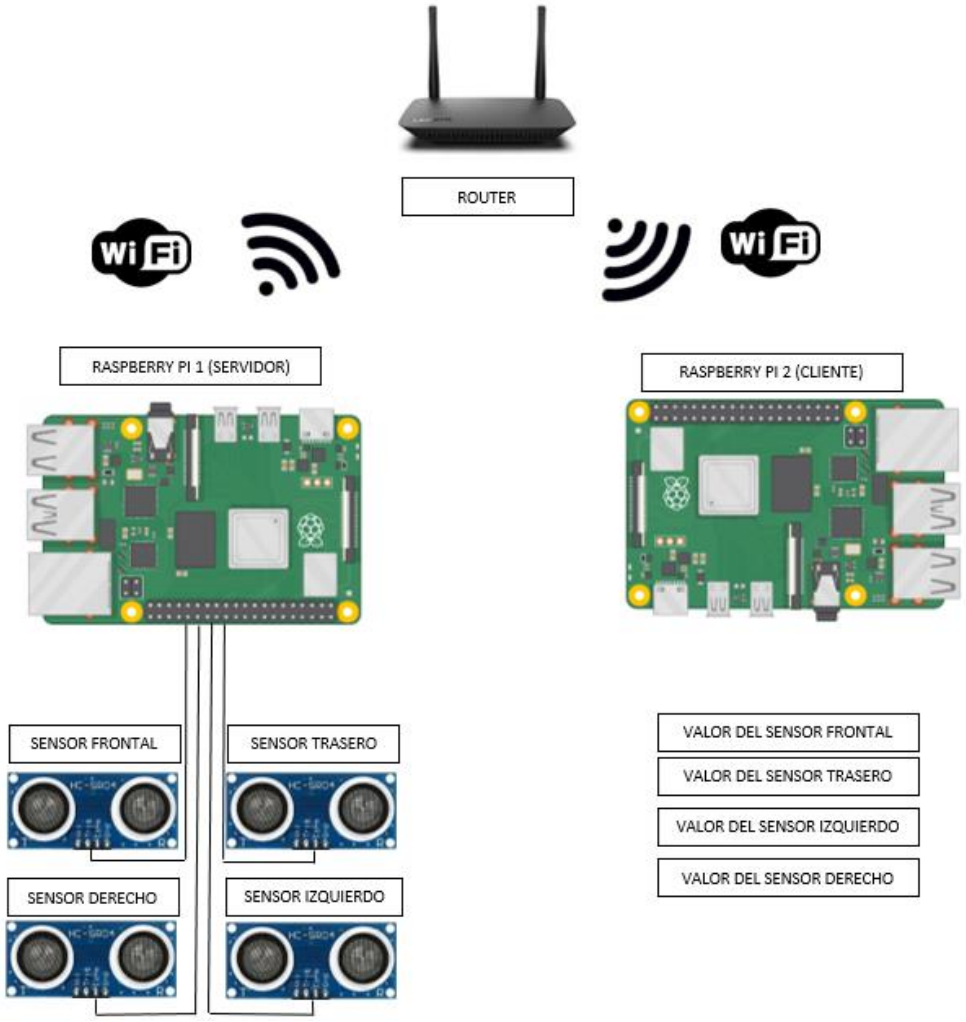

Figura 9. Servidor - Cliente TCP/IP

Luego de recibir los datos desde el servidor TCP/IP se realizaron 4 condiciones en las que se establece una distancia fija predefinida para que cuando una persona u objeto estén cerca del sensor, se encienda un círculo de color rojo, indicando una obstrucción. Estas condiciones se establecieron por cada sensor en tiempo real.

El código se puede ver en la sección de ANEXOS Código fuente para el cliente TCP/IP, al igual que su respectivo diagrama de bloques.

5. Geolocaliazación mediante GPS

Para el caso de la geolocalización, se utilizó el módulo UBLOX NEO 6, el cual mediante comunicación serial proporciona datos relativos a las coordenadas de ubicación. Cabe recalcar que el módulo UBLOX NEO 6 se conecta a los satélites GPS para transmitir la ubicación exacta, pero si la antena no tiene la suficiente cobertura, los datos van a ser erróneos y la ubicación no va a ser la correcta. En el presente proyecto se utilizaron las librerías serial pynmea2 de Python para establecer las conexiones y recolectar los datos. Para visualizar los datos de forma correcta se decodificaron los caracteres recibidos desde el puerto serial a Unicode utf-8. En la Figura 10 se muestran las conexiones realizadas.

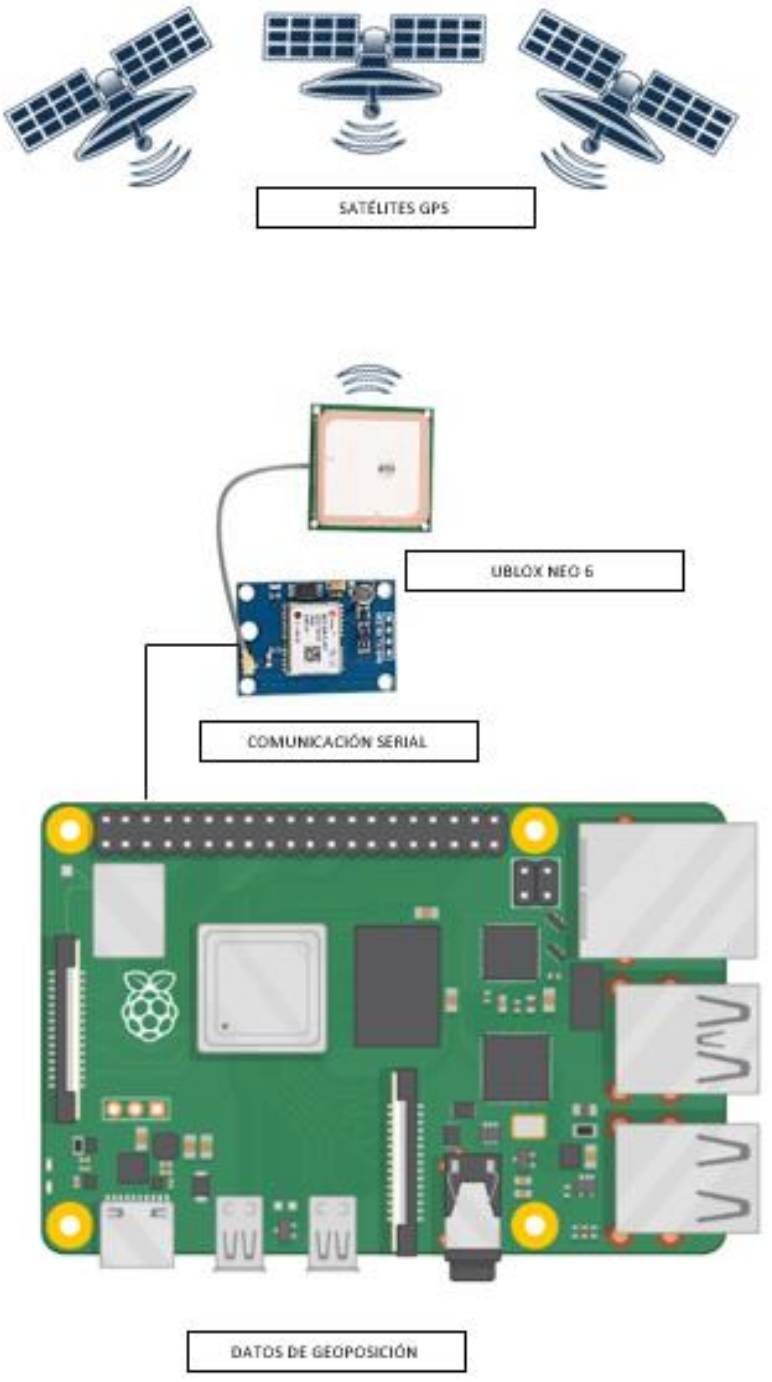

Figura 10. Geolocalización con GPS en Raspberry Pi 2

Una vez establecido el posicionamiento exacto de las coordenadas de longitud y latitud, se almacenan en dos variables independientes que se actualizan en tiempo real. Esto se lo realizó para ubicar en un mapa de google maps la ubicación a la cual pertenecen la latitud y longitud procesadas.

5.1 Código Fuente para la Geolocalización mediante GPS

```
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 pyrebase


def gps():
    global lat
    global lng
    global map_link
    while True:
        port = "/dev/ttyAMA0"
        ser = serial.Serial(port, baudrate=9600, timeout=0.5)
        dataout = pynmea2.NMEAStreamReader()
        newdata = ser.readline()
        n_data = newdata.decode('utf-8')
        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}
            map_link = 'http://maps.google.com/?q=' + str(lat) + ',' +
str(lng) #Link para abrir Google maps con la latitud y longtud
            print(gps)
```

Este código está presente en el segundo Raspberry en el código de cliente. En la sección de anexos se puede observar su respectivo diagrama de bloques.

6. Visualización de las Cámaras

Con la ayuda de la biblioteca opencv2 para Python se pudo establecer la apertura de las cámaras y usarlas para visualizar en tiempo real las imágenes proyectadas. Las cámaras utilizadas en el presente proyecto se colocaron en la parte

frontal y trasera del vehículo respectivamente. En la figura 11 se detallan las conexiones para las cámaras para el segundo Raspberry Pi .

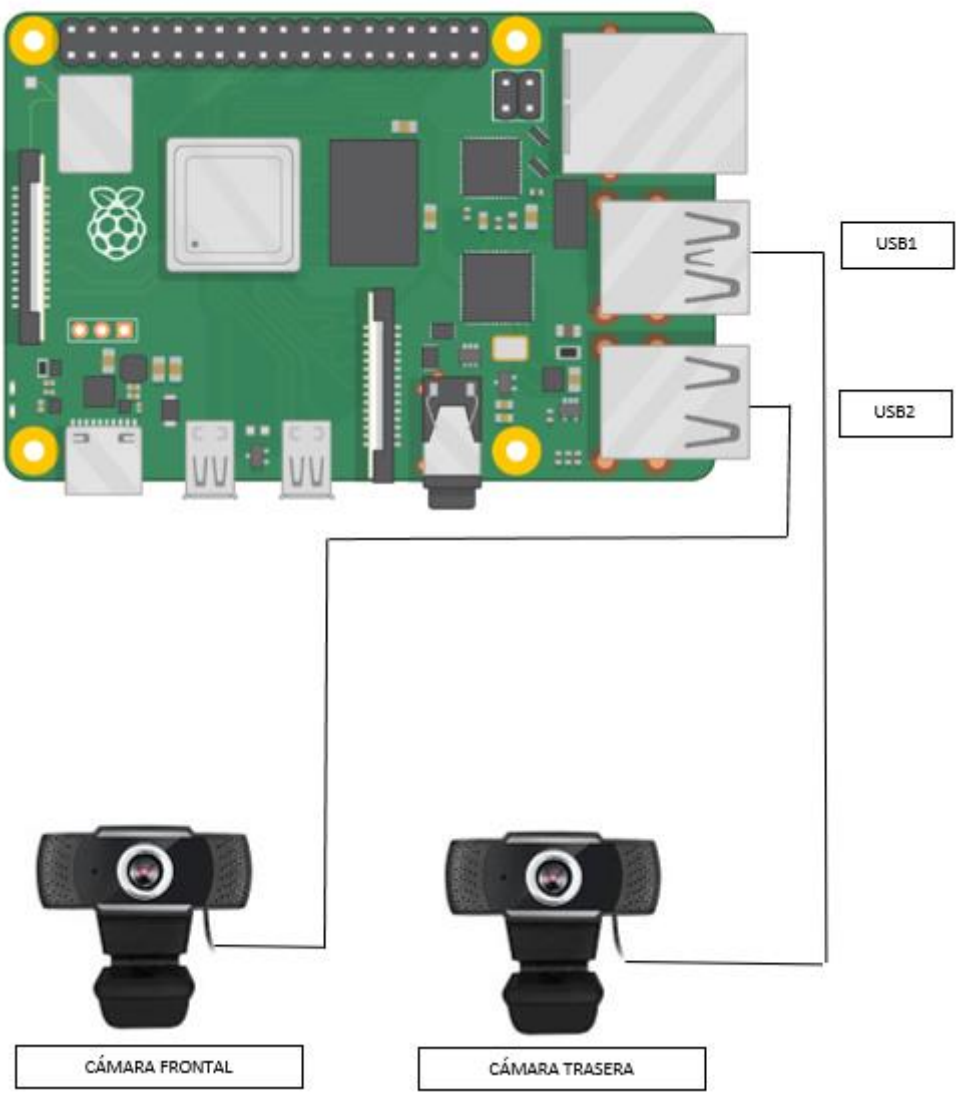

Figura 11. Conexión de cámaras hacia el segundo Raspberry Pi.

Para realizar esta configuración, primeramente se entra a la configuración del Raspberry en la ventana del terminal con sudo raspi-config para confirmar que la cámara este activada.

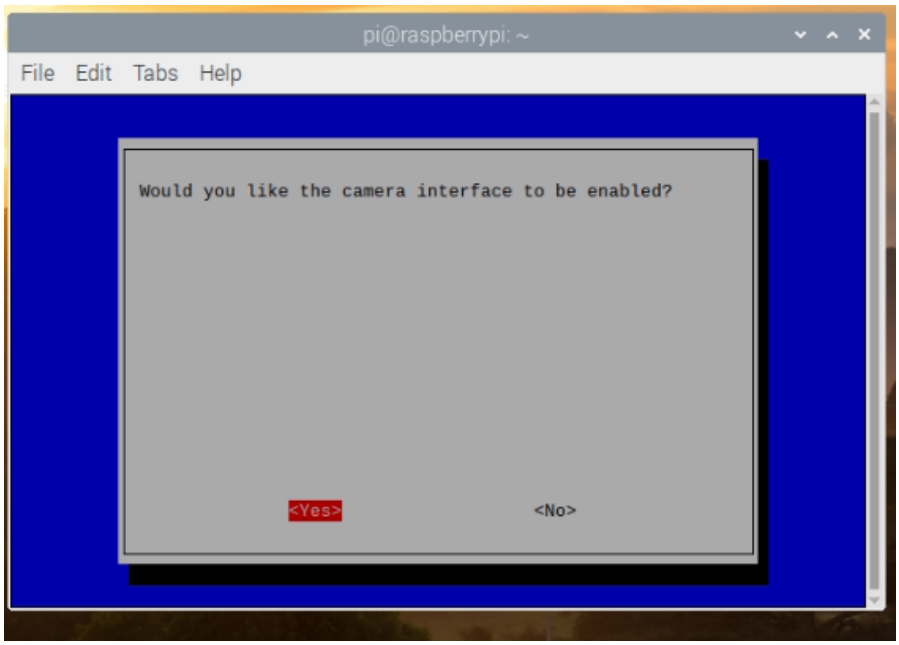

Figura 12. Habilitación de la cámara

En la Figura 13 se puede observar el espacio de interfaz que ocupa cada webcam conectada mediante USB mediante el comando, ls –la /dev/vid*. Así nos cercioramos de lo antes dicho para colocar los bits adecuados en el código que se verá más adelante. Por ejemplo se puede ver que la cámara 1 ocupa la interfaz 0 y 1 por esto en el códifgo se poner que la cámara ocupa el espacio de interfaz 0 (siempre se escoge el de menor valor). La cámara 2 ocupa el espacio de interfaz 2 y 3 por lo tanto en el código se puso que ocupa el espacio 2.

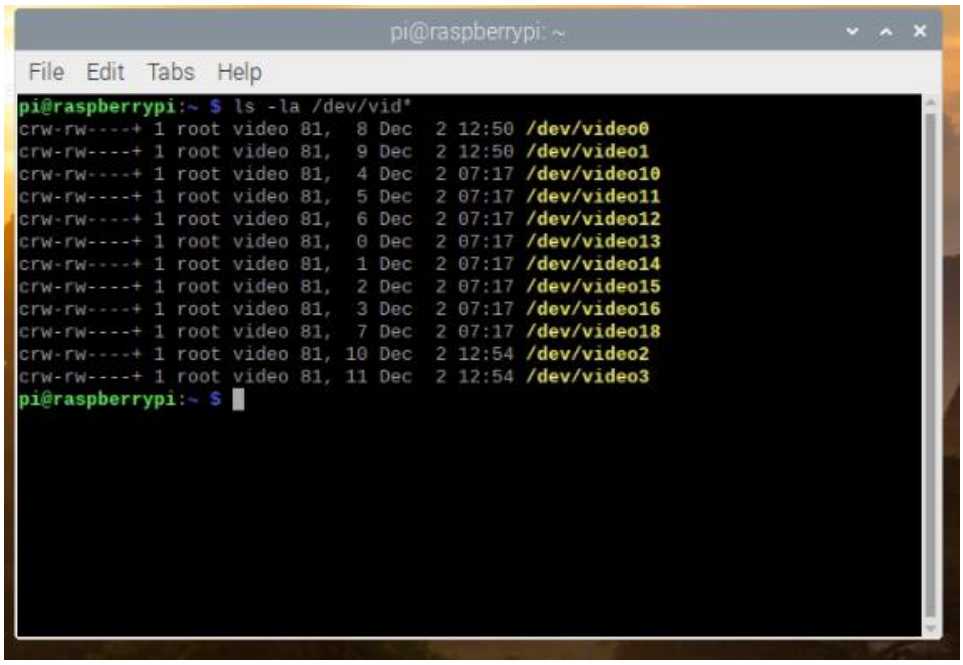

Figura 13. Puntos de conexión Cámaras

Se procede a escribir los siguientes comandos: Sudo apt-get install fswebcam y Fswebcam image.jpg, para comprobar que las cámaras están funcionando y nos proporcionan una imagen clara del entorno.

Después de las pruebas y verificaciones de funcionamiento y de que el Raspberry lee las cámaras se procede a la realización del código en Python encargado de monitorear las dos cámaras, de desplegar al mismo tiempo las dos imágenes del entorno.

El código que se muestra al final de esta sección realiza la importación de las librerías cv2 (opencv) y numpy, y luego declara las entradas de video y la función de captura de video. Después, en el ciclo while true, captura cuadro a cuadro, las imágenes que se visualizan en cada una de las cámaras. En los bloques if se retiene la información y posteriormente se muestra la captura de lo que proyecta el while true para las dos cámaras correspondientes. El último if es el comando de cierre del ciclo. Y los últimos comandos son para cerrar el video y las ventanas correspondientes y así poner fin al programa.

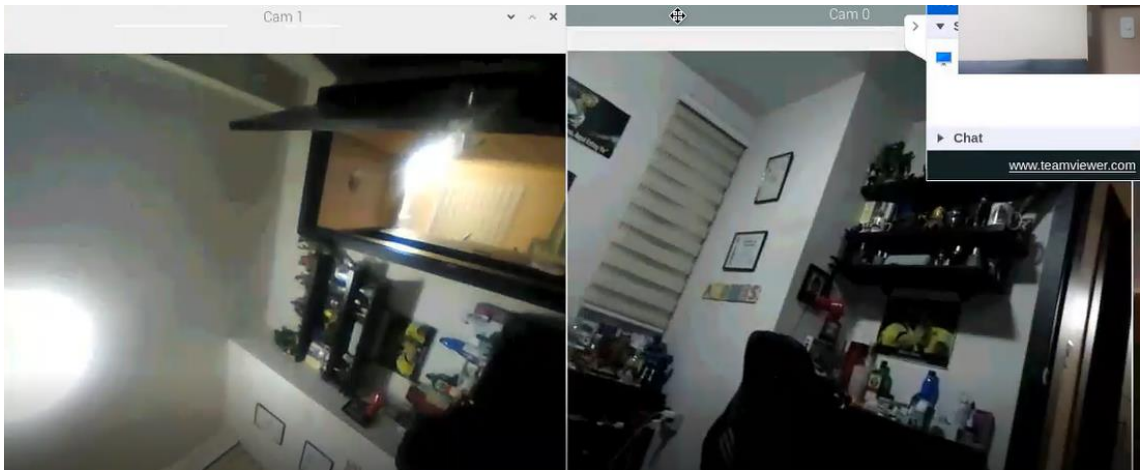

Figura 14. Despliegue de dos cámaras

En esta figura 14 podemos observar que el código de las cámaras antes explicado está funcionando de manera perfecta al desplegarnos una ventana con el enfoque de las dos cámaras que se procederán a usar en el carro.

Luego de esto se procede a la introducción del código a la interfaz gráfica con sus respectivas modificaciones. En la figura 15 se puede observar una ventana interactiva producida por la librería pygame en la cual el usuario puede visualizar el estado de los sensores, los datos del GPS, además de poder desplegar una pantalla con la ubicación exacta en espacio y tiempo por medio de Google Maps y como se ve ya está implementado las dos cámara dentro de la interfaz aquí se puedo verl el enfoque de las dos cámaras.

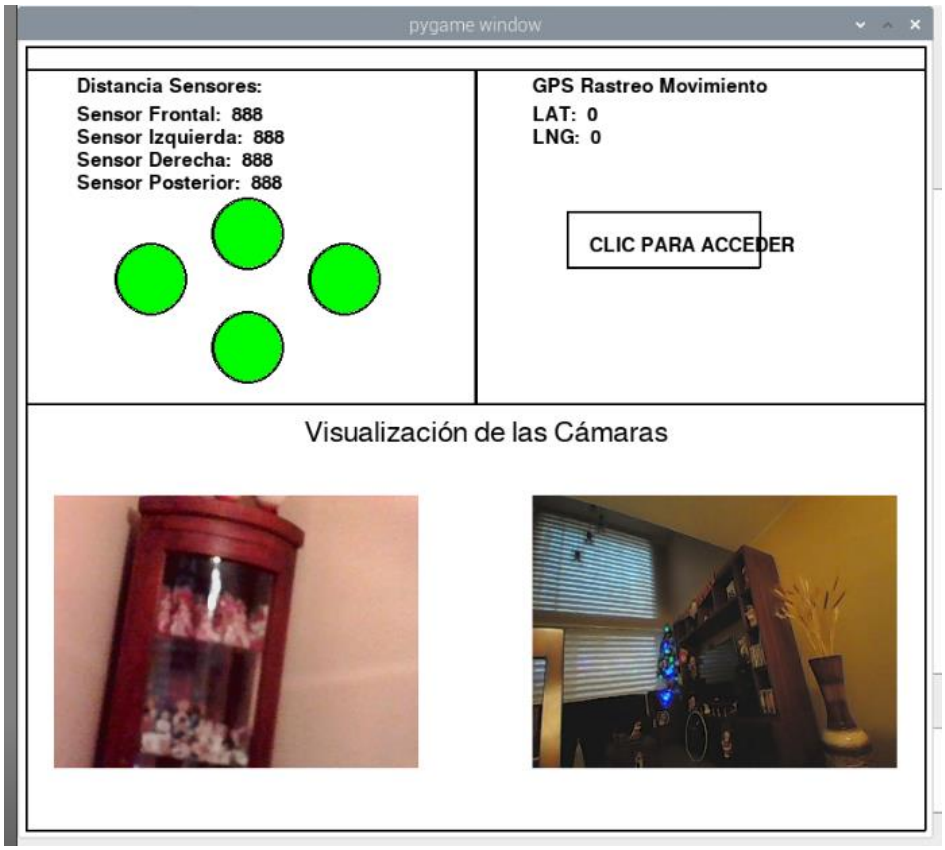

Figura 15. Interfaz implementado código cámaras

Para implementar las cámaras en la interfaz gráfica se procedió a realizar una restructuración del código. Primero se implementa las librerías a utilizar, se determina la resolución de la cámara que en este caso es 320x240 que debido al procesador del Raspberry pi es la resolución óptima para que el programa se ejecute de una manera óptima que es sin interferencias, de una forma veloz y con el espacio suficiente para que se enfoquen los dos cuadros de cámaras. Se definen las funciones camstream1 y camstream2, luego se crea un ciclo infinito con el while true en donde se iniciaría la librería pygame y se inicia las cámaras, acto seguido se configura el tamaño de la cámara y se inicia los recuadros en el eje x y eje y, en la posición 0. Se declara la variable camera 1 que es para elegir el dispositivo y el tamaño. Posteriormente se muestra la imagen que proyecta la cámara.

```
20  # Librerias para CAM1 y CAM2
 21 import numpy as np
 22 import cv2
 23
 24 # Variables para la CAM1 y CAM2
 25
 26 largo_cam = 320
 27 ancho_cam = 240
 28
 29 video_capture_0=cv2.VideoCapture(0)
 30 video_capture_1=cv2.VideoCapture(2)
 31
 32 video_capture_0.set(3,largo_cam)
 33 video_capture_0.set(4,ancho_cam)
 34
 35 video_capture_1.set(3,largo_cam)
 36 video_capture_1.set(4,ancho_cam)
 37
 38 #Fin de las variables para la CAM1 y CAM2

235 def camera():
236     while True:
237         # Capture frame-by-frame
238         ret0, frame0 = video_capture_0.read()
239         ret1, frame1 = video_capture_1.read()
240
241         if (ret0):
242           # Display the resulting frame
243             cv2.resizeWindow('Cam 0', largo_cam, ancho_cam)
244             cv2.imshow('Cam 0', frame0)
245             cv2.moveWindow('Cam 0', 1010,515)
246
247         if (ret1):
248          # Display the resulting frame
249             cv2.resizeWindow('Cam 1', largo_cam, ancho_cam)
250             cv2.imshow('Cam 1', frame1)
251             cv2.moveWindow('Cam 1', 590,515)
252
253         if cv2.waitKey(1) & 0xFF == ord('q'):
254             break
255     video_capture_0.release()
256     video_capture_1.release()
257     cv2.destroyAllWindows()
```

5. Interfaz Gráfica

En la segunda Rapsberry pi se tiene el código para unificar todos los datos y mostrarlos en una ventana, se utilizó la librería pygame, con la ayuda de esta librería se

construyeron: el fondo de la ventana, los círculos de los sensores indicadores de obstrucción de los sensores ultrasónicos, los datos de los sensores ultrasónicos, los datos de latitud y longitud, un bloque dinámico para abrir un mapa de Google maps y la visualización de las cámaras. Para la visualización de todos los datos, se utilizó hilos de procesamiento mediante la librería threading.

En la sección de anexos se puede observar su respectivo diagrama de bloques

5.1 Código Fuente para la Visualización de la interfaz Gráfica

En la segunda raspberry reside este código, es un código que está organizado en 3 secciones la primera es los sockets para la visualización de los datos de los sensores, la segunda es la sección de lectura de datos del GPS además de desplegar un mapa de la ubicación en tiempo real del segundo raspberry en google maps, y la sección de la visualización de las dos cámaras. Entonces por la librería pygame todos estos datos se unifican y se muestran en una ventana interactiva.

```
 5
 6 from pygame.constants import USEREVENT
 9 import pygame.locals
10 import serial
11 import time
12 import string
13 import pynmea2
50 import threading
51 WHITE = (255, 255, 255)
52 BLACK = (0, 0, 0)
53 GREEN = (0, 255, 0)
54 RED = (255, 0, 0)
55 BLUE = (0, 0, 255)
56
57 T1 = "Distancia Sensores:"
58 T2 = "Sensor Frontal:  "
59 T3 = "Sensor Izquierda:  "
60 T4 = "Sensor Derecha:  "
61 T5 = "Sensor Posterior:  "
62 T6 = "GPS Rastreo Movimiento"
63 T7 = "LAT:  "
64 T8 = "LNG:  "
65
```

```
pygameDisplay = pygame.display.set_mode((800, 850), 0, 32) #Ubicación de la interfaz


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, 690), 2) #Rectángulo externo 1
        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, RED, (5, 25, 790, 670), 2) # Rectágulo externo2
        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, 375), 2) # Rectángulo Cámaras
        pygame.display.update()

    def addText(self):
        self.screen.blit(self.font.render('Visualización de las Cámaras', True, BLACK), (250, 330)) # Ubicación del texto "Cámaras"
        pygame.display.update()
```

```
115
116 class Lower_Rect_C1(object):
117     def __init__(self):
118         pygame.init()
119         self.font = pygame.font.SysFont('Arial', 25)
120         self.screen = pygame.display.set_mode((800, 600), 0, 32)
121         self.screen.fill(WHITE)
122         pygame.display.update()
123
124     def addRect(self):
125         self.rect = pygame.draw.rect(self.screen, BLUE, (30, 400, 320,
126 240), 5) # Rectángulo Cámara1
127         pygame.display.update()
128
129 class Lower_Rect_C2(object):
130     def __init__(self):
131         pygame.init()
132         self.font = pygame.font.SysFont('Arial', 25)
133         self.screen = pygame.display.set_mode((800, 600), 0, 32)
134         self.screen.fill(WHITE)
135         pygame.display.update()
136
137
138     def addRect(self):
139         self.rect = pygame.draw.rect(self.screen, RED, (450, 400, 320,
140 240), 5) # Rectángulo Cámara2
141         pygame.display.update()
142
143 class Two_Rect_Left(object):
144     def __init__(self):
145         pygame.init()
146         self.font = pygame.font.SysFont('Calibre', 25)
147         self.screen = pygame.display.set_mode((800, 600), 0, 32)
148         self.screen.fill(WHITE)
149         pygame.display.update()
150
151     def addRect(self):
152         self.rect = pygame.draw.rect(self.screen, BLACK, (5, 25, 395,
153 295), 2) # Rectágulo Sensores
154         pygame.display.update()
155
156     def addText(self):
157         pygame.time.set_timer(pygame.USEREVENT, 1000)
158
159         sock0 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
160         sock1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
161         sock2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
162         sock3 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
163         server_address0 = ('192.168.168.205', 2000)
```

```
164         server_address1 = ('192.168.168.205', 2001)
165         server_address2 = ('192.168.168.205', 2002)
166         server_address3 = ('192.168.168.205', 2003)
167         sock0.connect(server_address0)
168         sock1.connect(server_address1)
169         sock2.connect(server_address2)
170         sock3.connect(server_address3)
171         data0 = sock0.recv(8)
172         data1 = sock1.recv(8)
173         data2 = sock2.recv(8)
174         data3 = sock3.recv(8)
175         distancia1 = float(data0.decode("utf-8"))
176         distancia2 = float(data1.decode("utf-8"))
177         distancia3 = float(data2.decode("utf-8"))
178         distancia4 = float(data3.decode("utf-8"))
179         self.screen.blit(self.font.render(T1, True, BLACK), (50, 30))
180         self.screen.blit(self.font.render(T2 + str(distancia1), True,
181 BLACK), (50, 55))
182         sock0.close()
183         self.screen.blit(self.font.render(T3 + str(distancia3), True,
184 BLACK), (50, 75))
185         sock1.close()
186         self.screen.blit(self.font.render(T4 + str(distancia2), True,
187 BLACK), (50, 95))
188         sock2.close()
189         self.screen.blit(self.font.render(T5 + str(distancia4), True,
190 BLACK), (50, 115))
191         sock3.close()
192         if distancia1 < dis_thresh:
193             pygame.draw.circle(self.screen, RED, (200, 170), 30, 0)
194             pygame.draw.circle(self.screen, BLACK, (200, 170), 32, 2)
195         else:
196             pygame.draw.circle(self.screen, GREEN, (200, 170), 30, 0)
197             pygame.draw.circle(self.screen, BLACK, (200, 170), 32, 2)
198         if distancia3 < dis_thresh:
199             pygame.draw.circle(self.screen, RED, (115, 210), 30, 0)
200             pygame.draw.circle(self.screen, BLACK, (115, 210), 32, 2)
201         else:
202             pygame.draw.circle(self.screen, GREEN, (115, 210), 30, 0)
203             pygame.draw.circle(self.screen, BLACK, (115, 210), 32, 2)
204         if distancia2 < dis_thresh:
205             pygame.draw.circle(self.screen, RED, (285, 210), 30, 0)
206             pygame.draw.circle(self.screen, BLACK, (285, 210), 32, 2)
207         else:
208             pygame.draw.circle(self.screen, GREEN, (285, 210), 30, 0)
209             pygame.draw.circle(self.screen, BLACK, (285, 210), 32, 2)
210         if distancia4 < dis_thresh:
211             pygame.draw.circle(self.screen, RED, (200, 270), 30, 0)
212             pygame.draw.circle(self.screen, BLACK, (200, 270), 32, 2)
```

```
213         else:
214             pygame.draw.circle(self.screen, GREEN, (200, 270), 30, 0)
215             pygame.draw.circle(self.screen, BLACK, (200, 270), 32, 2)
216         pygame.display.update()
217 
218 
219 class Two_Rect_Right(object):
220 
221     # Rectángulo blanco de fondo
222     def __init__(self):
223         pygame.init()
224         self.font = pygame.font.SysFont('Calibre', 25)
225         self.screen = pygame.display.set_mode((800, 700), 0, 32)
226         self.screen.fill(WHITE)
227         pygame.display.update()
228 
229     def addRect(self):
230         self.rect = pygame.draw.rect(self.screen, BLACK, (400, 25, 395,
231 295), 2) # Rectángulo GPS
232         pygame.display.update()
233 
234     def addText(self):
235         pygame.time.set_timer(pygame.USEREVENT, 1000)
236         self.screen.blit(self.font.render(T6, True, BLACK), (450, 30))
281         self.screen.blit(self.font.render(T7 + str(lat), True, BLACK),
282 (450, 55))
283         self.screen.blit(self.font.render(T8 + str(lng), True, BLACK),
284 (450, 75))
285         pygame.display.update()
286 
287     def addButton(self):
288 
289         self.rectButton = pygame.draw.rect(self.screen, BLACK, (480,
290 150, 225, 50), 2)
291         self.screen.blit(self.font.render('CLICK PARA ACCEDER', True,
292 BLACK), (500, 170))
293         pygame.display.update()
294 
295     def buttonClick(self):
296         if self.rectButton.collidepoint(pygame.mouse.get_pos()):
297 
298             webbrowser.open(map_link, new=2)
299 
300 if __name__ == '__main__':
301     t1 = threading.Thread(target=gps)
302     t1.start()
303     t2 = threading.Thread(target=camera)
304     t2.start()
305     MR = Main_Rect()
```

```
306     UR = Upper_Rect()
307     LR = Lower_Rect()
308     LRC1 = Lower_Rect_C1()
309     LRC2 = Lower_Rect_C2()
310     TRL = Two_Rect_Left()
311     TRR = Two_Rect_Right()
312     MR.addRect()
        UR.addRect()
        LR.addRect()
        LRC1.addRect()
        LRC2.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()
```

Conclusiones:

- El uso de dos webcams es mejor para el correcto procesamiento del Raspberry debido a que se probó con cámara del Raspberry que es una cámara de 8 megapixeles que es capaz de tomar imágenes estáticas 3280 x 2464 pixeles y capturan video a 1080p30, mismo y se tuvo un problema con la captura de video se hizo más lento el procesamiento de imágenes y no soportaba la resolución que tenía la cámara del Raspberry que era 640x480p90.

- El uso de Python es esencial para el desarrollo de este proyecto, al realizar todas las secciones del proyecto en este programa la lectura de los sensores, el GPS y la captura de video de las cámaras.

- Durante el desarrollo del proyecto se pudo observar que para el uso del módulo GPS se requiere establecer una comunicación serial, en donde se tienen dos señales digitales: una señal trasmisora y una señal receptora.

- También se pudo reconocer que existe un flujo de datos constante que envía el GPS que sirven para poder hacer un rastreo del movimiento que está haciendo el carrito. Este flujo de datos constante además del uso en vivo del video de las cámaras hace que se ponga más lento la interfaz gráfica.

- Luego de hacer uso de los sensores de proximidad, se llega a la conclusión que los sensores HCSR04 tiene una precisión muy buena y envían datos constantemente como también los hace el GPS.

- Los sockets son esenciales en el proceso de monitoreo de los sensores, debido a que en el carrito al haber dos programas que detectan los sensores ultrasónicos y miden su interacción va a haber una colisión de señales y solo uno de estos podrá mostrar los datos de los sensores. Al realizar los sockets los dos programas se conectan a un Access point y uno de esto va a actuar como server y otro como cliente, el server será el encargado de mandar los datos de los sensores al cliente y así los dos programas mostrarán los datos de los sensores.
- Fue mejor utilizar las cámaras USB porque el procesador del Raspberry trabaja mejor con este tipo de cámara y no se hace más lento además es mucho más cómodo a la hora de colocar las cámaras en el carro.
- Para la obtención de datos del GPS se debe estar en un lugar descubierto y alto, debido a que si se lo realiza en los laboratorios en los subsuelos no se va a tener señal del GPS.

Referencias:

- Barbero, J.M. y Rey, G. (1999). Los ejercicios del ver: Hegemonía audiovisual y ficción televisiva. Barcelona: Editorial Gedisa.
- Bonsiepe, G. (1998). Del objeto a la interface: Mutaciones del diseño. Buenos Aires: Ediciones Infinito Buenos Aires.
- Vujović, V., & Maksimović, M. (2014, May). Raspberry Pi as a Wireless Sensor node: Performances and constraints. In 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) (pp. 1013-1018). IEEE.
- Hunt, J. (2019). Sockets in Python. In Advanced Guide to Python 3 Programming (pp. 457-470). Springer, Cham.
- Saha, S., Singh, A., Bera, P., Kamal, M. N., Dutta, S., Gorian, U. & Sur, S. (2017, October). GPS based smart spy surveillance robotic system using Raspberry Pi for security application and remote sensing. In 2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON) (pp. 705-709). IEEE.
- Tashakkori, R., Hernandez, N. P., Ghadiri, A., Ratzloff, A. P., & Crawford, M. B. (2017, March). A honeybee hive monitoring system: From surveillance cameras to Raspberry Pis. In SoutheastCon 2017 (pp. 1-7). IEEE.
- Zholdangarova, G. I., Bolatkhan, A., Kozhakhmetov, R., & Baktybai, I. (2020). TCP Protocol in Python. In Вестник научных конференций (No. 2-3, pp. 11-13). ООО Консалтинговая компания Юком.

ANEXOS:

Los Anexos son partes complementarias de lo visto en el desarrollo del proyecto.

1.1 Código fuente para el servidor TCP/IP

```
#! /usr/bin/env python
#versionControlManualAuto version beta
import pygame
import RPi.GPIO as GPIO
import time
import numpy as np
import socket
import sys
# Creación del servidor TCP/IP
sock0 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock3 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Establecimiento de los puertos TCP/IP 2000,2001,2002 y 2003 locales.
server_address0 = ('', 2000)
server_address1 = ('', 2001)
server_address2 = ('', 2002)
server_address3 = ('', 2003)

print('starting up on {} port {}'.format(*server_address0))
print('starting up on {} port {}'.format(*server_address1))
print('starting up on {} port {}'.format(*server_address2))
print('starting up on {} port {}'.format(*server_address3))
sock0.bind(server_address0)
sock1.bind(server_address1)
sock2.bind(server_address2)
sock3.bind(server_address3)

# Escuchando las conexiones TCP/IP con un máximo de 5 clientes.
sock0.listen(5)
sock1.listen(5)
sock2.listen(5)
sock3.listen(5)


#pines sensores Proximidad


TRIG_F=3  #pin 16
TRIG_A=5  #pin 15
TRIG_I=7 #pin 13
TRIG_D=11 # pin 11


```

```
 66
 67 ECHO_F =37  #pin18
 68 ECHO_A=35 # pin 22
 69 ECHO_I=31  #pin 22
 70 ECHO_D=29 #pin 29
 74 # Definicion del modo GPIO
 75
 76 GPIO.setmode(GPIO.BOARD)
 77
 78 GPIO.setwarnings(False)
 79 GPIO.setup(INA1,GPIO.OUT)
 80 GPIO.setup(INB1,GPIO.OUT)
 81 GPIO.setup(INA2,GPIO.OUT)
 82 GPIO.setup(INB2,GPIO.OUT)
 83 GPIO.setup(PWM1,GPIO.OUT)
 84 GPIO.setup(PWM2,GPIO.OUT)
 85
 86 GPIO.setwarnings(False)
 87 GPIO.output(PWM1,GPIO.HIGH)
 88 GPIO.output(PWM2,GPIO.HIGH)
 89 GPIO.output(INA1,GPIO.LOW)
 90 GPIO.output(INB1,GPIO.LOW)
 91 GPIO.output(INA2,GPIO.LOW)
 92 GPIO.output(INB2,GPIO.LOW)
 93
 94 GPIO.setup(VCCLED,GPIO.OUT)
 95 GPIO.output(VCCLED,GPIO.LOW)
 96
 97 GPIO.setwarnings(False)
 98 freq=100
 99 p1=GPIO.PWM(PWM1,freq)
100 pwm1=30
101 p1.start(pwm1)
102
103 GPIO.setwarnings(False)
104 freq=100
105 p2=GPIO.PWM(PWM2,freq)
106 pwm2=55
107 p2.start(pwm2)
108
109
110 #def GPIO pines sensores
111
112 GPIO.setup(TRIG_F,GPIO.OUT)
113 GPIO.setup(TRIG_D,GPIO.OUT)
114 GPIO.setup(TRIG_I,GPIO.OUT)
115 GPIO.setup(TRIG_A,GPIO.OUT)
116
117 GPIO.setup(ECHO_F,GPIO.IN)
```

```
118 GPIO.setup(ECHO_D,GPIO.IN)
119 GPIO.setup(ECHO_I,GPIO.IN)
120 GPIO.setup(ECHO_A,GPIO.IN)
121
122 GPIO.output(TRIG_F, False)
123 GPIO.output(TRIG_D, False)
124 GPIO.output(TRIG_I, False)
125 GPIO.output(TRIG_A, False)
126
127 def VCCLUCESON():
128     GPIO.output(VCCLED,True)
129
130 def VCCLUCESOFF():
131     GPIO.output(VCCLED,False)
132
133 def distanciaFrente():
134     connection0, client_address0 = sock0.accept()#Socket 1
135     GPIO.output(TRIG_F, True)
136     time.sleep(0.00001)
137     GPIO.output(TRIG_F, False)
138
139     while GPIO.input(ECHO_F) ==0:
140
141         global pulse_startf
142
143         pulse_startf = time.time()
144
145
146     while GPIO.input(ECHO_F)==1:
147
148         global pulse_endf
149
150         pulse_endf = time.time()
151
152     pulse_durationf = pulse_endf - pulse_startf
153
154     distancef = pulse_durationf * 17150
155
156     distancef = round(distancef, 2)
157
158       #Envío de la señal del sensor frontal
159     signal0 = str(distancef)
160     data0 = signal0.encode("utf-8")
161     connection0.sendall(data0)
162
163
164     return distancef
165
166
```

```
167
168 def distanciaDerecha():
169     connection1, client_address1 = sock1.accept()#Socket 2
170     GPIO.output(TRIG_D, True)
171     time.sleep(0.00001)
172     GPIO.output(TRIG_D, False)
173
174     while GPIO.input(ECHO_D)==0:
175           global pulse_start
176           pulse_start = time.time()
177
178     while GPIO.input(ECHO_D)==1:
179           global pulse_end
180           pulse_end = time.time()
181
182     pulse_duration = pulse_end - pulse_start
183
184     distance = pulse_duration * 17150
185 #Envío de la señal del sensor derecho
186     distance = round(distance, 2)
187     signal1 = str(distance)
188     data1 = signal1.encode("utf-8")
189     connection1.sendall(data1)
194     return distance
195 def distanciaIzquierda():
196      connection2, client_address2 = sock2.accept()#Socket 3
197      GPIO.output(TRIG_I, True)
198      time.sleep(0.00001)
199      GPIO.output(TRIG_I, False)
200
201      while GPIO.input(ECHO_I)==0:
202
203            global pulse_start3
204            pulse_start3 = time.time()
205
206      while GPIO.input(ECHO_I)==1:
207
208            global pulse_end3
209            pulse_end3 = time.time()
210
211      pulse_duration3 = pulse_end3 - pulse_start3
212
213      distance3 = pulse_duration3 * 17150
214
215      distance3 = round(distance3, 2)
216      #Envío de la señal del sensor izquierdo
217      signal2 = str(distance3)
218      data2 = signal2.encode("utf-8")
219      connection2.sendall(data2)
```

```
224      return distance3
225
226 def distanciaAtras():
227     connection3, client_address3 = sock3.accept()#Socket 4
228     GPIO.output(TRIG_A, True)
229
230     time.sleep(0.00001)
231     GPIO.output(TRIG_A, False)
232
233     while GPIO.input(ECHO_A)==0:
234           global pulse_starta
235           pulse_starta = time.time()
236
237     while GPIO.input(ECHO_A)==1:
238           global pulse_enda
239           pulse_enda = time.time()
240
241     pulse_durationa = pulse_enda - pulse_starta
242
243     distancea = pulse_durationa * 17150
244
245     distancea = round(distancea, 2)
246
247     #Envío de la señal del sensor trasero
248     signal3 = str(distancea)
249     data3 = signal3.encode("utf-8")
250     connection3.sendall(data3)
251
252
253
254     return distancea
```

2.1 Código fuente para el cliente TCP/IP

```
 14 #Librerías para los sockets
 15 import socket
 16 import sys
 17
 18 # Librerias para CAM1 y CAM2
 19 import numpy as np
 20 import cv2
 21
 36 distance_list= [888,888,888,888]
 37 dis_thresh = 5.00
 38
 42 WHITE = (255, 255, 255)
 43 BLACK = (0, 0, 0)
 44 GREEN = (0, 255, 0)
 45 RED = (255, 0, 0)
 46 BLUE = (0, 0, 255)
 47
 48 T1 = "Distancia Sensores:"
 49 T2 = "Sensor Frontal:  "
 50 T3 = "Sensor Izquierda:  "
 51 T4 = "Sensor Derecha:  "
 52 T5 = "Sensor Posterior:  "

148     def addText(self):
149         pygame.time.set_timer(pygame.USEREVENT, 1000)
150         #Conexión hacia el servidor remoto
151         sock0 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
152         sock1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
153         sock2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
154         sock3 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
155         server_address0 = ('192.168.168.205', 2000)
156         server_address1 = ('192.168.168.205', 2001)
157         server_address2 = ('192.168.168.205', 2002)
158         server_address3 = ('192.168.168.205', 2003)
159         sock0.connect(server_address0)
160         sock1.connect(server_address1)
161         sock2.connect(server_address2)
162         sock3.connect(server_address3)
163         data0 = sock0.recv(8)
164         data1 = sock1.recv(8)
165         data2 = sock2.recv(8)
166         data3 = sock3.recv(8)
167         distancia1 = float(data0.decode("utf-8"))
168         distancia2 = float(data1.decode("utf-8"))
169         distancia3 = float(data2.decode("utf-8"))
170         distancia4 = float(data3.decode("utf-8"))
```

```
171         self.screen.blit(self.font.render(T1, True, BLACK), (50, 30))
172         self.screen.blit(self.font.render(T2 + str(distancia1), True,
173 BLACK), (50, 55))
174         sock0.close()
175         self.screen.blit(self.font.render(T3 + str(distancia3), True,
176 BLACK), (50, 75))
177         sock1.close()
178         self.screen.blit(self.font.render(T4 + str(distancia2), True,
179 BLACK), (50, 95))
180         sock2.close()
181         self.screen.blit(self.font.render(T5 + str(distancia4), True,
182 BLACK), (50, 115))
183         sock3.close()
184         if distancia1 < dis_thresh:
185             pygame.draw.circle(self.screen, RED, (200, 170), 30, 0)
186             pygame.draw.circle(self.screen, BLACK, (200, 170), 32, 2)
187         else:
188             pygame.draw.circle(self.screen, GREEN, (200, 170), 30, 0)
189             pygame.draw.circle(self.screen, BLACK, (200, 170), 32, 2)
190         if distancia3 < dis_thresh:
191             pygame.draw.circle(self.screen, RED, (115, 210), 30, 0)
192             pygame.draw.circle(self.screen, BLACK, (115, 210), 32, 2)
193         else:
194             pygame.draw.circle(self.screen, GREEN, (115, 210), 30, 0)
195             pygame.draw.circle(self.screen, BLACK, (115, 210), 32, 2)
196         if distancia2 < dis_thresh:
197             pygame.draw.circle(self.screen, RED, (285, 210), 30, 0)
198             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()
```

Diagrama de Bloques Servidor TCP

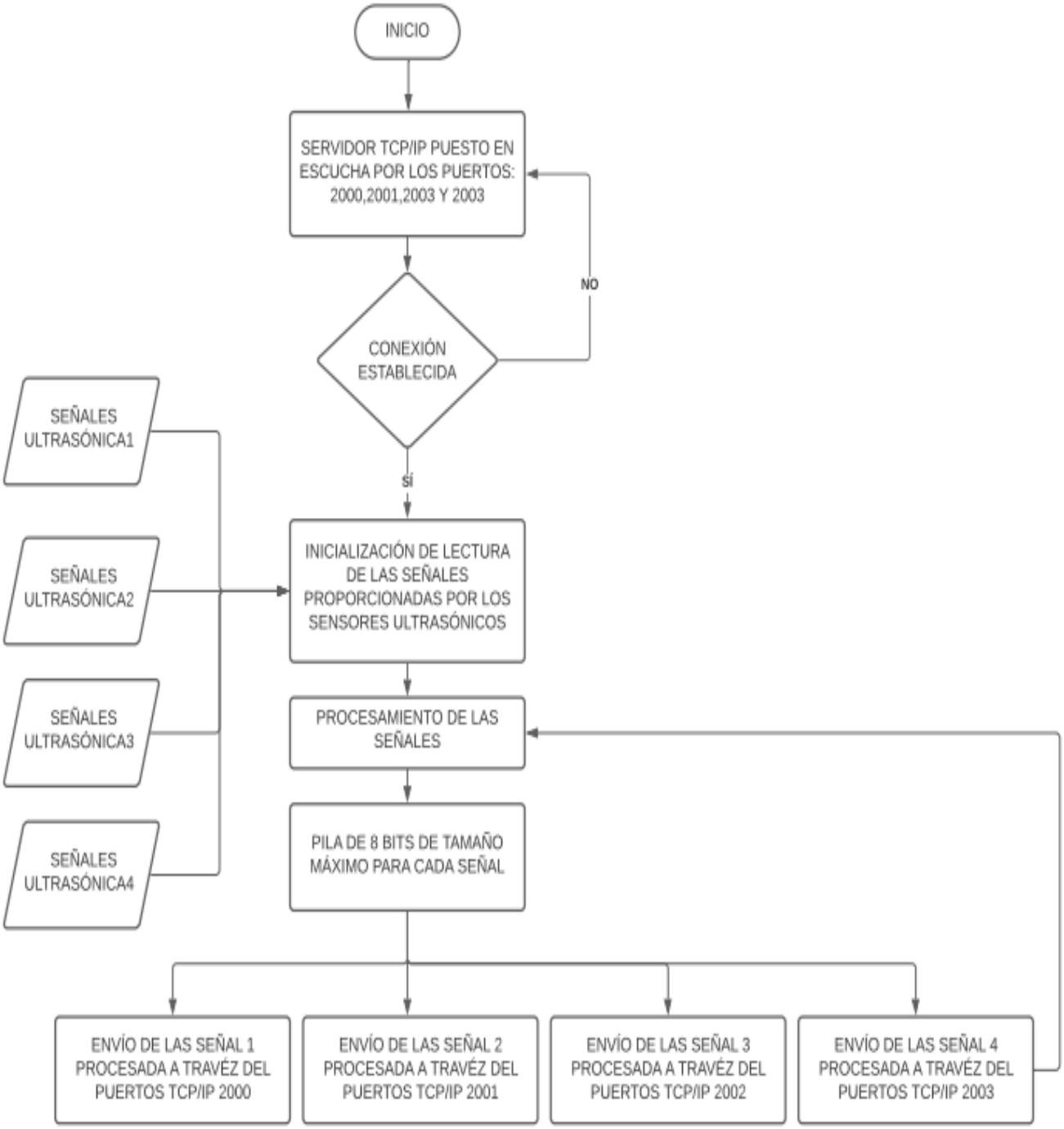

Diagrama de Bloques Cliente TCP

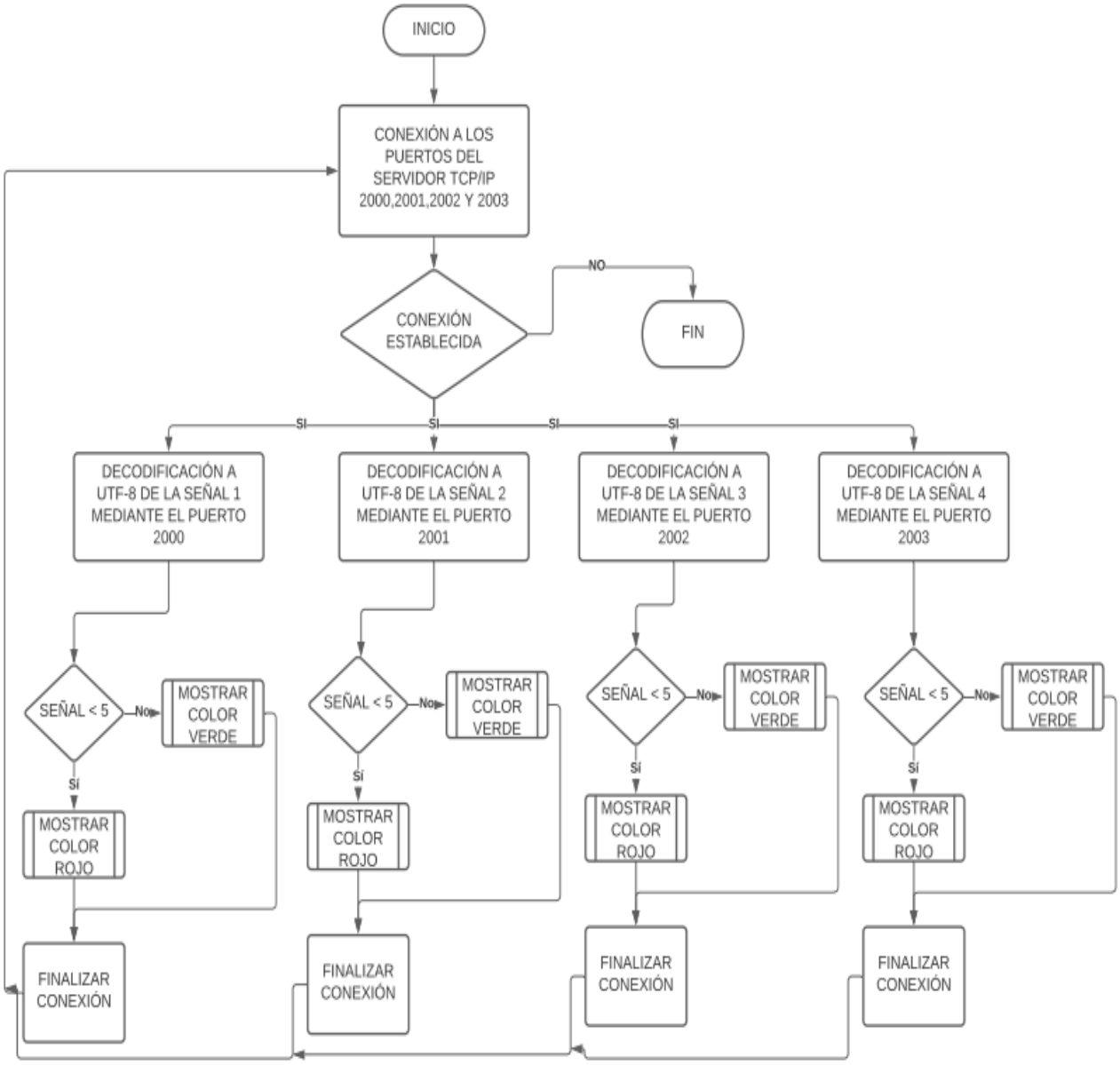

Diagrama de Bloques GPS

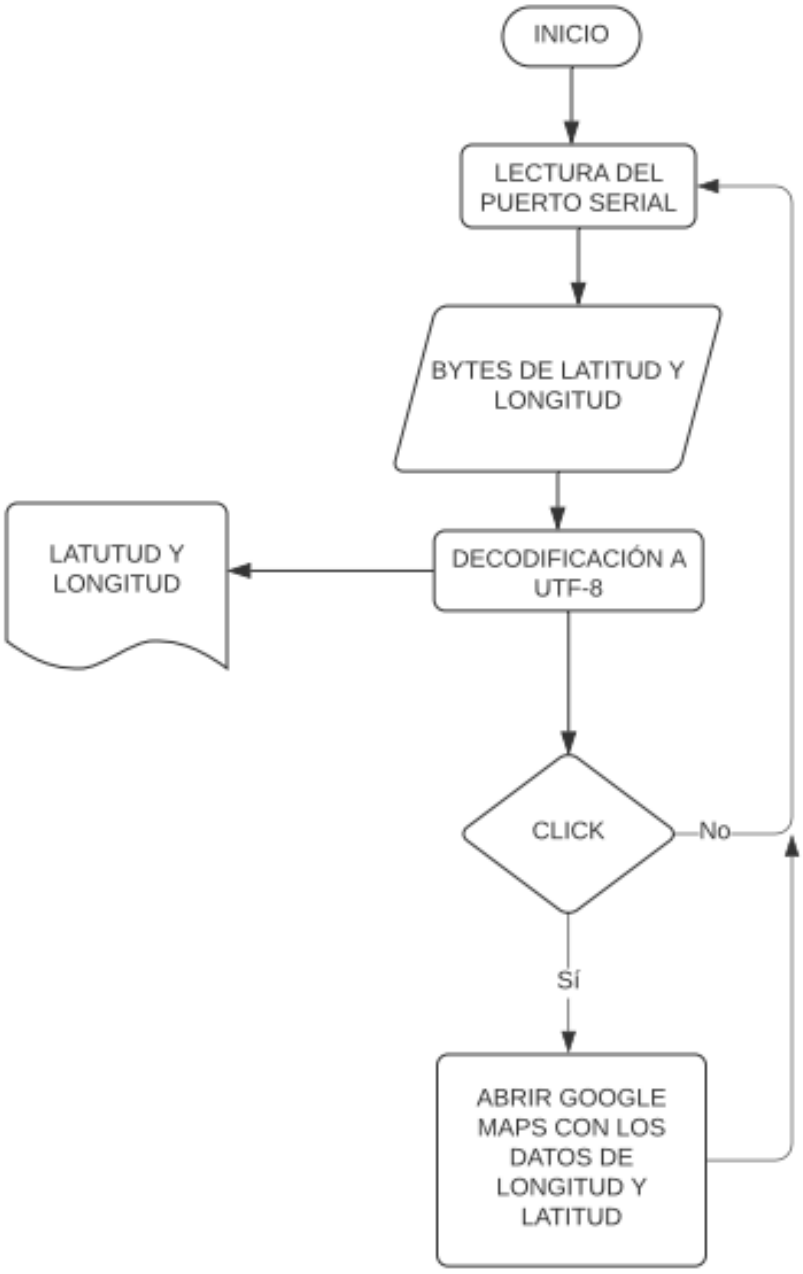

Diagrama de Bloques Cámaras