Implementación de un sistema APRS utilizando LoRa - iGate

Alberto González Guardia
Escuela de Ingeniería Electrónica
Cartago, Costa Rica
alberto@estudiantec.cr

Ivan Campbell Tames
Escuela de Ingeniería Electrónica
Cartago, Costa Rica
ivancampbell@estudiantec.cr

Daniela Vargas Chavarria
Escuela de Ingeniería Electrónica
Cartago, Costa Rica
danielavargas0407@estudiantec.cr

***Abstract*—This report details the configuration process required for a gate in an APRS system to receive information and retransmit it to web pages, explains the first level block diagram, details the code used, explains the complications when implementing said code and how they were solved, and suggests an application in which this type of system can be used along with its prototype.**

Key words – APRS, Firmware, Gate, LoRa, Tracker.

I. Introducción

Un sistema APRS (Automatic Packet Reporting System) permite intercambiar de manera rápida información en tiempo real por medio de una red de estaciones que utilizan bandas y frecuencias para transmisión de datos [1]. Entre la información que puede transmitirse se encuentra la ubicación GPS (si se cuenta con un modulo GPS), información meteorológica si se hace uso de sensores, mensajes de texto, etc.

Estos sistemas son comúnmente utilizados con LoRa (Long Range), esta tecnología utiliza un tipo de modulación de amplio espectro, la cual permite que el ruido no causa una gran interferencia [2]. Gracias a esto se puede dar la comunicación inalámbrica a larga distancia.

Para que se de la transmisión se necesitan dos dispositivos diferentes, trackers y gateways, los trackers son los encargados de enviar la información, cuentan con un modulo GPS que permite saber cual es su ubicación cuando esta en movimiento, esto hace que sean útiles en aplicaciones de rastreo de autos o de emergencia. Por otro lado, los gateways permiten recibir los datos para posteriormente retransmitirlos a, por ejemplo, un pagina web.

En el caso de este proyecto, unicamente se hizo uso del gateway (gate), por lo tanto, el enfoque se dara a esta parte del sistema APRS.

II. Metodología

En esta sección se explica la metodología utilizada para configurar un gate LILYGO. Primeramente, siguiendo con lo indicado en el diseño modular se diseñan los diagramas de bloques, en la figura 1 se muestra el diagrama de bloques de primer nivel.

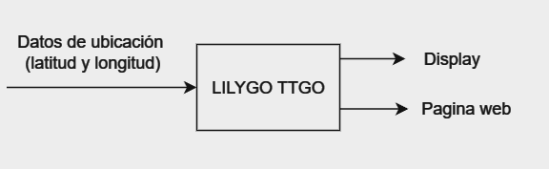

Figura 1: Diagrama de bloques de primer nivel.

Como se puede observar, la entrada del diagrama son los datos de ubicación y las salidas son el display y una pagina web que se encuentra en el siguiente hipervínculo: https://aprs.fi/#!lat=9.93460&lng=-84.07070, esta pagina es APRS.fi, en esta se muestran los trackers y gates en el mapa y es posible visualizar los datos. Sin embargo, para que el gate pueda conectarse con un tracker y mostrar los datos en el display es necesario configurarlo utilizando un firmware.

Este se carga en el gate utilizando Visual Studio Code, desde el cual se descarga platformio, el cual permite programar embebidos como el gate. El código utilizado se muestra en la figura 2 y en la figura 3.

Figura 2: Código utilizado para configurar el gate (Parte 1).

```
{
    "callsign": "TiTEC3-10",
    "wifi": {
        "AP": [
          {
            "ssid": "honor",
            "password": "dani0707"
          }
        ],
        "autoAP": {
          "password": "dani0707",
          "powerOff": 10
        }
    },
    "beacon": {
        "latitude": 9.8516581,
        "longitude": -83.9097920,
        "comment": "LoRa APRS Grupo3 - Taller Integrador II Semestre",
        "interval": 15,
        "overlay": "/",
        "symbol": "[",
        "path": "WIDE1-1",
        "sendViaAPRSIS": true,
        "sendViaRF": true
    },
    "digi": {
        "mode": 0
    },
    "tnc": {
        "enableServer": false,
        "enableSerial": false,
        "acceptOwn": false
    },
    "aprs_is": {
        "active": true,
        "passcode": "12509",
        "server": "noam.aprs2.net",
        "port": 14580,
        "filter": "m/10",
        "toRF": true
    },
```


Figura 3: Código utilizado para configurar el gate (Parte 2).

El primer parámetro importante es el 'callsign', este es el identificador que tiene el gate, en este caso TiTEC3-10. Posteriormente se configura la red WiFi a la que se va a conectar el dispositivo, es importante considerar que la red debe operar en la banda de 2,4 GHz.

Posteriormente se ingresan los datos correspondientes a la latitud y longitud, en este caso esos valores corresponden a 9.8516581 para la latitud y -83.997920 para la longitud, esta corresponde a una ubicación en Cartago. Ademas de esto se agrega un comentario para que posteriormente este aparezca en la pagina de APRS. Otro aspecto a tomar en cuenta es el 'passcode' el cual en este caso es 12509, este es único y se asocia uno a cada callsign.

Una vez configurado el código se corre y se carga en el gate, como se muestra en la figura 4.

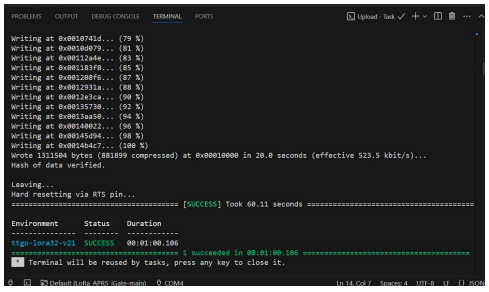

Figura 4: Código cargado en el gate

Después de configurarlo, es posible diseñar una aplicación para este, en este caso se sugiere un sistema de rastreo para camiones de carga, en la figura 5 y en la figura 6 se muestran los diagramas de tercer y quinto nivel respectivamente, donde se explica este diseño.

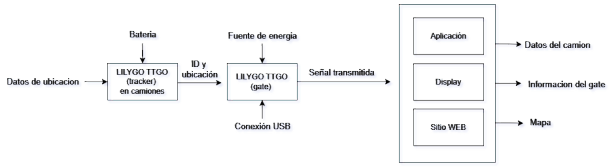

Figura 5: Diagrama de tercer nivel

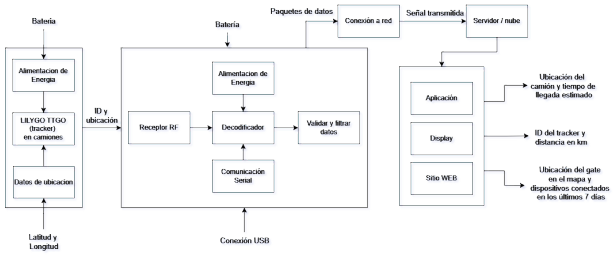

Figura 6: Diagrama de quinto nivel

Con la configuración explicada anteriormente es posible obtener dos de las salidas que se visualizan en el diagrama, que son los datos del display y del mapa en la pagina web, sin embargo, se requiere de otro código para poder obtener la tercer salida, que corresponde a una aplicación.

El código utilizado se muestra en la sección 'Apéndice', este corresponde a un código realizado con Python, donde inicialmente se ingresa a la pagina de APRS y se descarga los datos del gate y del tracker, en este caso los datos de ubicación son los requeridos para posteriormente calcular la distancia entre ambos utilizando una formula llamada Haversina, la cual permite calcular la distancia en kilómetros utilizando la diferencia entre la latitud y la longitud [4]. Esta ecuación también se muestra en la sección 'Apéndice'.

Posteriormente se implementa una función que genera una interfaz para poder visualizar datos como los identificadores y la distancia calculada.

III. Análisis de resultados

En la figura 7 se muestra el gate antes de ser configurado de manera correcta.

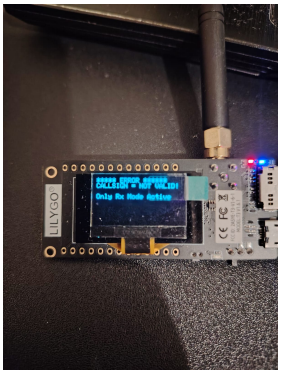

Figura 7: Código cargado de manera incompleta en el gate

En este caso, se había cargado la configuración sin embargo no se habían cargado der manera correcta los datos sobre la red del internet lo cual hacia que apareciera el error mostrado en la imagen.

En la figura 8 se muestra el gate una vez cargado el código de manera correcta, se puede observar como en el display aparece el callsign TiTEC3-10, esta conectado de manera correcta a una red WiFi, el IP, el numero de estaciones cerca y que se ha conectado al tracker TI3WTI-10 y que este se encuentra a 0.8km.

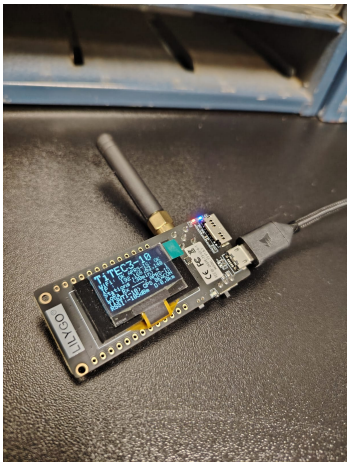

Figura 8: Código cargado en el gate correctamente [Ejemplo 1]

Posteriormente, en la figura 9 se muestra otro ejemplo, en este caso las coordenadas fueron modificadas para probar que se pudiera visualizar en el display el cambio en la distancia, por ese motivo en este caso la distancia con el mismo tracker del ejemplo anterior es de 0.5 km.

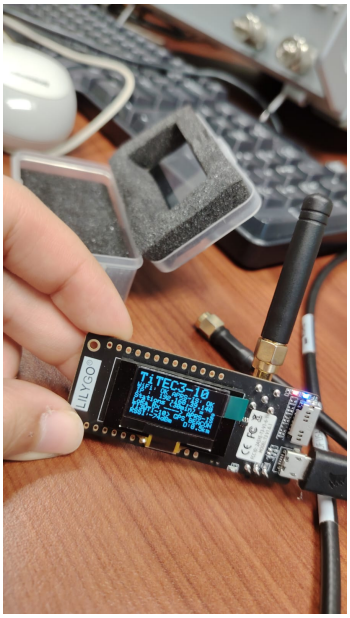

Figura 9: Código cargado en el gate [Ejemplo 2]

En la figura 10 se muestra el gate visto en la pagina https://aprs.fi/, se puede observar como aparece el callsign del código mostrado en la figura 2, ademas de un comentario y la fecha en la que se configuro para que apareciera en el mapa de la pagina de aprs.

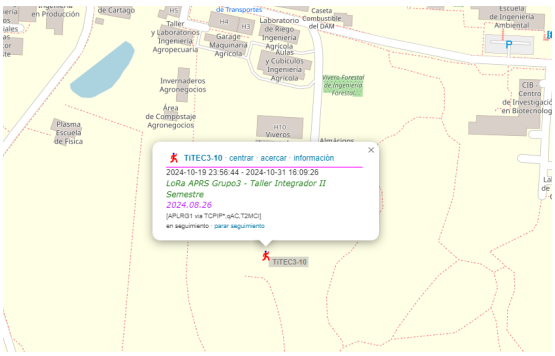

Figura 10: Código cargado en el gate

Seleccionando la opción de 'Información' es posible visualizar mas datos , como los mostrados en la figura 11.

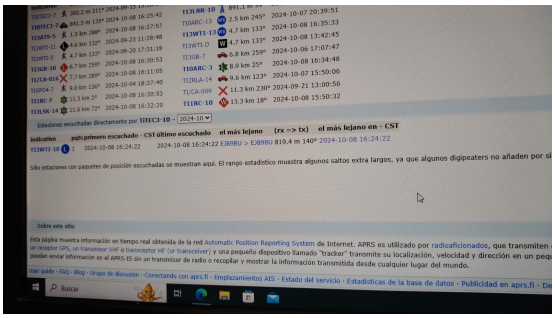

Figura 11: Código cargado en el gate

En este se puede observar a cuales dispositivos se ha conectado en los últimos siete días, ademas de la fecha y distancia a la que se encontraban. Estos datos también pueden visualizarse desde la aplicación creada, en la figura 12 se muestra la interfaz.

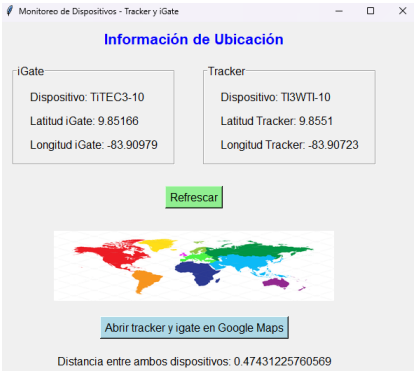

Figura 12: Código cargado en el gate

Como se muestra en la imagen se observan los datos de los dispositivos conectados, en la izquierda se muestra el

identificador del gate, junto con sus coordenadas y en la derecha el tracker junto con su identificador y coordenadas, se observa un botón para refrescar datos y otro para abrir google maps y ver la distancia desde ahí, como se muestra en la figura 13.

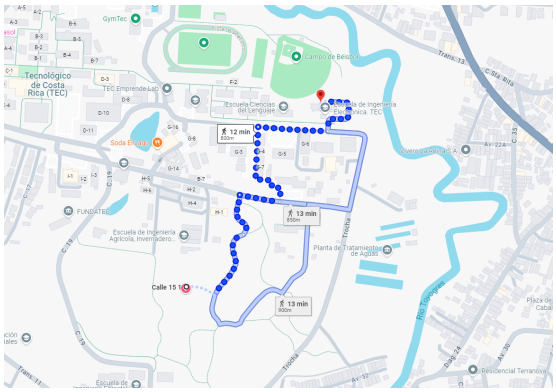

Figura 13: Código cargado en el gate

Además, se muestra la distancia entre ambos, en este caso es de 0.47 km, lo cual coincide con lo visto en el display de la figura 9.

IV. Conclusiones

La tecnología LoRa es sumamente útil para proyectos de IoT, en este caso se dio un prototipo de un proyecto en el cual su uso es para rastrear camiones de carga, sin embargo, tiene muchas aplicaciones donde puede ser útil en ciudades inteligentes y en la agricultura. Ademas, con sistemas APRS la transferencia de datos se da de manera rápida y efectiva.

Es necesario entender las diferencias entre un *Tracker* y entre un *iGate* ya que se configuran de forma distinta, y son componentes diferentes, los cuales tienen sus funciones diferentes. De entender bien sus diferencias y funcionamiento, se podría continuar desarrollando la tecnología y buscando aplicaciones prácticas que se le puedan sumar al prototipo.

Como consideración para investigación adicional, si se desea continuar desarrollando el producto con un énfasis en vender, en un sistema comercial, es fundamental que sea escalable y de fácil uso. Esto implica que el sistema debe ser plug-and-play, permitiendo que el usuario conecte el dispositivo sin realizar configuraciones iniciales, como ajustar coordenadas o modificar código. Cada unidad debe estar preconfigurada para minimizar el esfuerzo del usuario, garantizando que el dispositivo esté listo para operar de inmediato tras la conexión

Referencias

[1] CRECJ, 'Que es y cómo funciona el sistema APRS', 2024. [Online] Available: https://crecj.org/que-es-y-como-funciona-el-sistema-aprs/#:~:text=El%20APRS%20es%20un%20sistema%20de%20radioaficionados%20creado,bandas%20y%20frecuencias%20asignadas%20para%20transmisi%C3%B3n%20de%20datos.

[2] VENCO, 'Qué es LoRa, cómo funciona y características principales'. 2022. [Online] Available: https://www.vencoel.com/que-es-lora-como-funciona-y-caracteristicas-principales/

[3] National Instruments. An Introduction to Software Defined Radio. [Online] Available: https://forums.ni.com/t5/Community-Documents/Introduction-to-Software-Defined-Radio-with-NI-USRP-1-Hour-Hands/ta-p/3529496

[4] AcademiaLab.Fórmula de Haversina [Online] Available: https://academia-lab.com/enciclopedia/formula-de-haversina/

APÉNDICE

Listing 1: Código para aplicacion APRS con LoRa

```
import urllib.request
import json
import math
import tkinter as tk
from tkinter import font
from PIL import Image, ImageTk
import requests
from io import BytesIO
import webbrowser

def fetch_aprs_data_gate(callsign, api_key):
    url = "http://api.aprs.fi/api/get?name
    =TiTEC3-10&what=loc&format=JSON&apikey=
    204772.m27dxOL5tGRsaW"
    f = urllib.request.urlopen(url)
    file =  f.read()
    json_data = json.loads(file)
    lat_g = json_data['entries'][0]['lat']
    long_g = json_data['entries'][0]['lng']
    calls_g = json_data['entries'][0]['srccall']
    return lat_g, long_g, calls_g

def fetch_aprs_data_tracker(callsign, api_key):
    url = "http://api.aprs.fi/api/get?name=TI3WTI-10&what=loc&format=JSON&apikey=204772.m27dxOL5tGRsaW"
    f = urllib.request.urlopen(url)
    file =  f.read()
    json_data = json.loads(file)
    lat_t = json_data['entries'][0]['lat']
    long_t = json_data['entries'][0]['lng']
    calls_t = json_data['entries'][0]['srccall']
    return lat_t, long_t, calls_t

def haversine(lat1, lon1, lat2, lon2):
    R = 6371
    dlat = math.radians(lat2 - lat1)
    dlon = math.radians(lon2 - lon1)
    a = (math.sin(dlat / 2) ** 2 + math.cos(math.radians(lat1)) * math.cos(math.radians(lat2)) *  (math.
        sin(dlon / 2) ** 2))
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    distance = R * c
    return distance

data_g = fetch_aprs_data_gate("TiTEC3-10", "204772.m27dxOL5tGRsaW")
print(data_g)

data_t = fetch_aprs_data_tracker("TI3WTI-10", "204772.m27dxOL5tGRsaW")
print(data_t)

lat_g, lon_g, calls_g = data_g
lat_t, lon_t, calls_t = data_t

distance = haversine(float(lat_t), float(lon_t), float(lat_g),float(lon_g))
print("Distance from",calls_g,"to",calls_t, "is",distance,"km")

#para boton de refrescar datos
def refresh_window():
    data_g = fetch_aprs_data_gate("TiTEC3-10", "204772.m27dxOL5tGRsaW")
    data_t = fetch_aprs_data_tracker("TI3WTI-10", "204772.m27dxOL5tGRsaW")

    value4_label.config(text="Latitud Tracker: " + str(data_t[0]))
    value5_label.config(text="Longitud Tracker: " + str(data_t[1]))
    tracker_name_label.config(text="Dispositivo: " + str(data_t[2]))

    value1_label.config(text="Latitud iGate: " + str(data_g[0]))
    value2_label.config(text="Longitud iGate: " + str(data_g[1]))
    igate_name_label.config(text="Dispositivo: " + str(data_g[2]))

    distancia_dispositivos = haversine(float(data_t[0]), float(data_t[1]), float(data_g[0]), float(
        data_g[1]))
    distance_value.config(text="Distancia entre ambos dispositivos: " + str(distancia_dispositivos))
```

```
69
70  #funcion para tomar coordenadas y mostrarlas en un mapa
71  def open_google_maps():
72      igate_coords = f"{data_g[0]},{data_g[1]}"
73      tracker_coords = f"{data_t[0]},{data_t[1]}"
74
75      # URL para Google maps
76      map_url = f"https://www.google.com/maps/dir/{igate_coords}/{tracker_coords}"
77
78      # Abrir URL
79      webbrowser.open(map_url)
80
81
82  def main():
83      # Create the main window
84      window = tk.Tk()
85      window.title("Monitoreo de Dispositivos - Tracker y iGate")
86      window.geometry("600x500")  # Set window size to be larger
87
88      # Define fonts
89      title_font = font.Font(family="Helvetica", size=16, weight="bold")
90      label_font = font.Font(family="Helvetica", size=12)
91
92      # Add a main title
93      title_label = tk.Label(window, text="Informaci n de Ubicaci n", font=title_font, fg="blue")
94      title_label.grid(row=0, column=0, columnspan=2, pady=10)
95
96      # Create frames to organize the data for iGate and Tracker
97      igate_frame = tk.LabelFrame(window, text="iGate", font=label_font, padx=10, pady=10)
98      tracker_frame = tk.LabelFrame(window, text="Tracker", font=label_font, padx=10, pady=10)
99
100     igate_frame.grid(row=1, column=0, padx=20, pady=10, sticky="nsew")
101     tracker_frame.grid(row=1, column=1, padx=20, pady=10, sticky="nsew")
102
103     # iGate data labels
104     global value1_label, value2_label, igate_name_label
105     igate_name_label = tk.Label(igate_frame, text="Dispositivo: " + data_g[2], font=label_font)
106     value1_label = tk.Label(igate_frame, text="Latitud iGate: " + str(data_g[0]), font=label_font)
107     value2_label = tk.Label(igate_frame, text="Longitud iGate: " + str(data_g[1]), font=label_font)
108
109     # Tracker data labels
110     global value4_label, value5_label, value6_label, tracker_name_label
111     tracker_name_label = tk.Label(tracker_frame, text="Dispositivo: " + data_t[2], font=label_font)
112     value4_label = tk.Label(tracker_frame, text="Latitud Tracker: " + str(data_t[0]), font=label_font)
113     value5_label = tk.Label(tracker_frame, text="Longitud Tracker: " + str(data_t[1]), font=label_font)
114
115     #Distancia entre ambos
116     global distance_value
117     distancia_dispositivos = haversine(float(data_t[0]), float(data_t[1]), float(data_g[0]), float(
            data_g[1]))
118     distance_value = tk.Label(text="Distancia entre ambos dispositivos: " + str(distancia_dispositivos),
             font=label_font)
119
120     # Place iGate labels in the iGate frame
121     igate_name_label.grid(row=0, column=0, padx=10, pady=5, sticky="w")
122     value1_label.grid(row=1, column=0, padx=10, pady=5, sticky="w")
123     value2_label.grid(row=2, column=0, padx=10, pady=5, sticky="w")
124
125     # Place Tracker labels in the Tracker frame
126     tracker_name_label.grid(row=0, column=0, padx=10, pady=5, sticky="w")
127     value4_label.grid(row=1, column=0, padx=10, pady=5, sticky="w")
128     value5_label.grid(row=2, column=0, padx=10, pady=5, sticky="w")
129
130     # Create a button to refresh the window
131     refresh_button = tk.Button(window, text="Refrescar", command=refresh_window, font=label_font, bg="
            lightgreen")
132     refresh_button.grid(row=2, column=0, columnspan=2, pady=20)
133
134     # Load and display the image from a URL at the bottom
135     url = "https://images.creativemarket.com/0.1.0/ps/1363862/4554/2832/m1/fpnw/wm1/world-map-blue-
            colors-.jpg?1465889604&s=739de0fe71a085f4f6d87f3fe01fd6a1"  # Replace with your image URL
136     response = requests.get(url)
137     img_data = response.content
138     image = Image.open(BytesIO(img_data))  # Load the image from the response content
```

```
    image = image.resize((400, 100), Image.Resampling.LANCZOS)  # Resize the image to fit
    photo = ImageTk.PhotoImage(image)

    image_label = tk.Label(window, image=photo)
    image_label.grid(row=3, column=0, columnspan=2, pady=10)

    image_label.image = photo

    map_button = tk.Button(window, text="Abrir tracker y igate en Google Maps", command=open_google_maps
        , font=label_font, bg="lightblue")
    map_button.grid(row=4, column=0, columnspan=2, pady=10)

    distance_value.grid(row=5, column=0, columnspan=2, pady=10)

    window.mainloop()

if __name__ == "__main__":
    main()
```

$$d = 2R \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\theta_2 - \theta_1}{2}\right) + \cos(\theta_1) \cdot \cos(\theta_2) \cdot \sin^2\left(\frac{\phi_2 - \phi_1}{2}\right)}\right) \tag{1}$$