IBM NALAIYA THIRAN 2022-23 PROJECT REPORT
SIGNS WITH SMART CONNECTIVITY FOR BETTER ROAD SAFETY
TEAM ID - PNT2022TMID35873

1 INTRODUCTION

1.1 PROJECT OVERVIEW

This project aims to replace static signboards with smart connected sign boards that can receive speed limits from a web application using weather API and update them automatically based on the weather, define diversion routes through API, and alert cars to school and hospital zones.

1.2 PROJECT DESCRIPTION

- To replace the static signboards, smart connected sign boards are used.
- These smart connected sign boards get the speed limitations from a web app using weather API and update automatically.
- Based on the weather changes the speed may increase or decrease.
- Based on the traffic and fatal situations the diversion signs are displayed.
- Guide(Schools), Warning and Service(Hospitals, Restaurant) signs are also displayed accordingly.
- Different modes of operations can be selected with the help of buttons.

2 LITERATURE SURVEY

2.1 EXISTING PROBLEM

- Normal Signboards need to be updated according to the needs.
- The nearby places are not informed to the driver to take appropriate steps.

2.2 REFERENCES

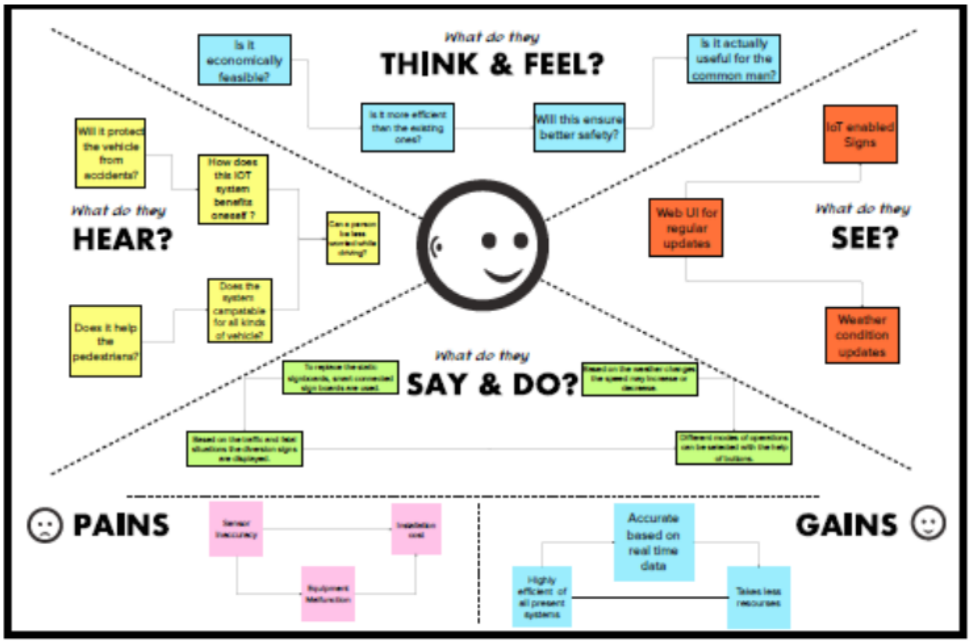
- Dariusz Grabowski & Andrzej Czyzewski in their paper titled "System for monitoring road slippery based on CCTV cameras and convolutional neural networks", Springer Publications 2020, made use of Convolutional Neural Networks to identify slippery roads using CCTV cameras.
- L.F.P. Oliveira, L.T. Manera, P.D.G. Luz in their paper titled "Smart Traffic Light Controller System", IEEE 2019, developed smart traffic lights capable of traffic accident detection enabling the enhancement of traffic light management systems, blocking and creating alternative routes to not only avoid the traffic jams, but also avoid new accidents.
- Muhammed O. Sayin, Chung-Wei Lin, Eunsuk Kang, Shinichi Shiraishi & Tamer Basar in their paper titled "Reliable Smart Road Signs", IEEE 2019, proposed a game theoretical adversarial intervention detection mechanism for reliable smart road signs. A future trend in intelligent transportation systems is “smart road signs” that incorporate smart codes (e.g., visible at infrared) on their surface to provide more detailed information to smart vehicles.

2.3 PROBLEM STATEMENT

To replace static signboards with smart connected signboards that can configure diversion routes using an API, alert vehicles to school and hospital zones, and get speed limit updates automatically based on the weather from a web app using weather API.

3 IDEATION AND PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS



3.2 BRAINSTORMING AND IDEATION

3.3 PROPOSED SOLUTION

The OpenWeatherMap API is used to receive the weather and temperature information. The speed limit will be automatically changed based on these information and the current weather. Additionally, information is gathered on any incidents and traffic jams that may have occurred on the specific road. On the basis of this, the traffic is diverted, the map's path is changed, and the traffic is then cleared. In order to make the traffic sign board more generic, additional buttons will be included. Each button will have a specific function, such as changing the warning signs, which are predefined and appear separately for the school and hospital zones.

4 REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENTS

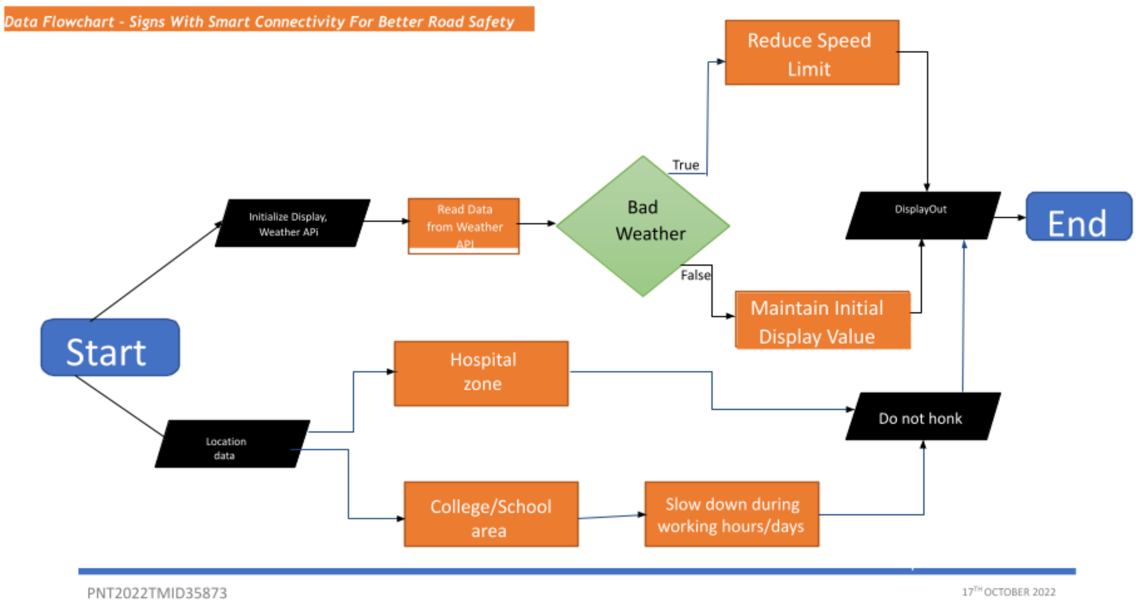
FR No.	Functional Requirement	Sub-Requirement
FR-1	**Convenience**	The implemented device must not be a distraction to the usual driving experience
FR-2	**Understanding**	Information Conveyed must be understandable by anyone driving the vehicle
FR-3	**Visibility**	Traffic and Weather Conditions must not be a hindrance to the signs

4.2 NON-FUNCTIONAL REQUIREMENTS

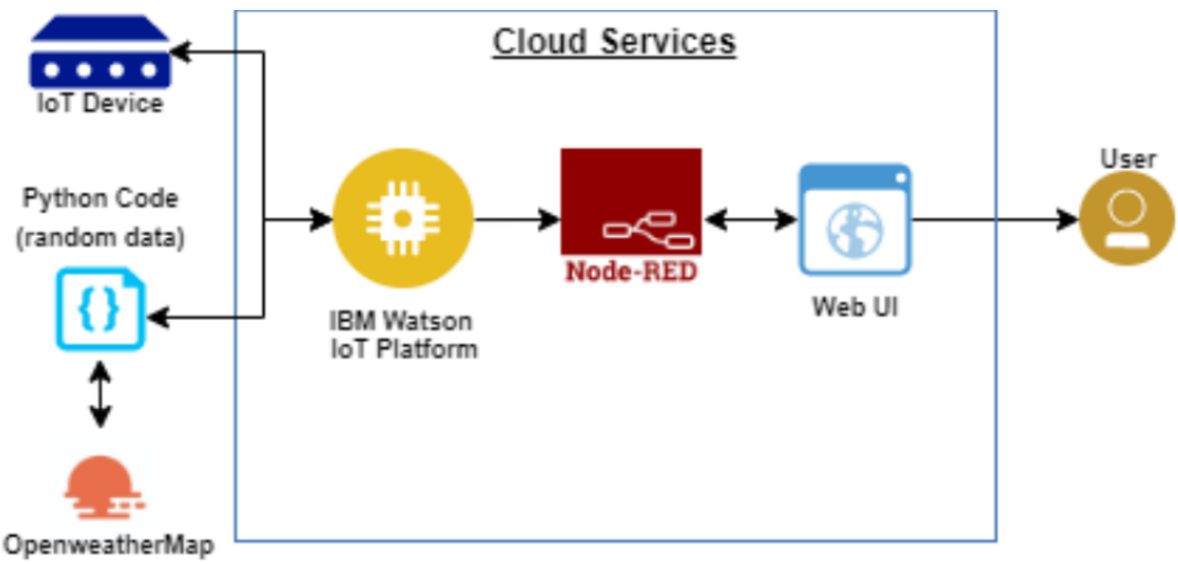
NFR No.	Non-Functional Requirement	Description
NFR-1	**Security**	Only intended data must be displayed and collected.
NFR-2	**Capacity**	Low Storage requirements, physically and memory-wise.
NFR-3	**Compatibility**	Suitable for all types of vehicles
NFR-4	**Reliability and Availability**	Must work on all conditions
NFR-5	**Maintainability and Manageability**	Easy to manage by anyone and service must be quick
NFR-6	**Scalability**	Must be able to manage huge workloads
NFR-7	**Usability**	Easy to use despite the environmental conditions

5 PROJECT DESIGN

5.1 DATA FLOW DIAGRAM



5.2 TECHNICAL ARCHITECTURE



6 PROJECT PLANNING AND SCHEDULING

6.1 SPRINT SCHEDULE

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Initialization of All required Services	USN-1	Initialize services like OpenWeatherMap, Node-RED, etc.	2	High	Keerthika R Abdul Raheem G A Jeffin Rohith J Kudumala Uday Kumar Reddy Harriet Perdita P
Sprint-2	Implementation of Code	USN-2	Coding to integrate all services as one	2	High	
Sprint-3	Hardware Integration	USN-3	Hardware implementation on IoT Enabled Device	2	Medium	
Sprint-4	Optimization	USN-4	Bug fixes and improvements	2	Low	

6.2 SPRINT DELIVERY PLAN

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	14 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	18 Nov 2022

7 CODING AND SOLUTIONING

7.1 FEATURE 1

DYNAMIC SPEED LIMITS:

- The standard speed limit is fixed and then according to the environmental variables the speed limit gets altered.
- Based on the visibility and the chances of rain, the speed limit is reduced to half to prevent accidents.

7.2 FEATURE 2

NEARBY LOCATION DETECTOR:

- The Sign board can detect the nearby places if the GPS Co-ordinates are entered
- It is programmed to filter out the nearby schools and hospitals and update the signboard as such, thereby preventing the driver to honk at a school or hospital zone.

8 TESTING

8.1 TEST CASES

Test Case 1	Clear Weather – Usual Speed Limit
Test Case 2	Unclear Weather – Reduced Speed Limit
Test Case 3	School/Hospital Zone – Do not Honk

8.2 USER ACCEPTANCE TESTING

User can avoid traffic and enjoy a safe trip home with the help of dynamic speed and diversion modifications dependent on the weather and traffic. The users would be in favour of this concept being used worldwide.

9 RESULTS

9.1 PERFORMANCE METRICS

The functionality of the website changes depending on the IBM bundle we selected. Node RED can process up to 10,000 requests per second and is based on NodeJS, a lightweight and high-performance engine. Moreover, a bigger demand of clients can be satisfied because the system is horizontally scalable.

10 ADVANTAGES AND DISADVANTAGES

10.1 ADVANTAGES

- More affordable and low-requirement microcontrollers can be used.
- More durable systems.
- Dynamic sign updates.
- Alerts for the School/Hospital Zones

10.2 DISADVANTAGES

- Display consumes more power.
- Dependent on API which is dependent on the servers, so it is vulnerable to a server blackout.

11 CONCLUSION

At a far reduced cost, our project can take the place of static signs, and it can be put into use right away. This will lessen many accidents and provide a calmer traffic environment across the nation.

12 FUTURE SCOPE

By giving the end-user (vehicle driver) the most precise information about the present road and traffic circumstances, the introduction of intelligent road sign groupings in real-life scenarios could have a significant impact on boosting driving safety. Smoother traffic flows and, more importantly, a greater driver awareness of the road situation could arise from even the simplest of information displays, such as a suggested driving speed and the condition of the road surface (temperature, icy, wet, or dry surface).

13 APPENDIX

1) GitHub Project Link:

https://github.com/IBM-EPBL/IBM-Project-4545-1658734390

2) Source Code:

a) weatherdata.py

```
import requests
# from time import sleep

#Function Definition
def get(myLocation,APIKEY):
    print("Connecting to OpenWeatherMap API")
    apiURL = f"https://api.openweathermap.org/data/2.5/weather?q={myLocation}&appid={APIKEY}"
    responseJSON = (requests.get(apiURL)).json()
    # print(responseJSON)
    returnObject = {
        "temperature" : responseJSON['main']['temp'] - 273.15,
        "weather" : [responseJSON['weather'][_]['main'].lower() for _
inrange(len(responseJSON['weather']))],
        "visibility" : responseJSON['visibility']/1000, # visibility in percentage where 10km is 100% and 0km
is 0%
        "wind" : responseJSON['wind']['speed'],
        "wind_dir" : responseJSON['wind']['deg']
    }
    print("Connected to OpenWeatherMap API")
    if("rain"in responseJSON):
        returnObject["rain"] = [responseJSON["rain"][key] for key in responseJSON["rain"]]
    return(returnObject)

#Testing
# while True:
# print(get("Chennai","c132fedc6afa3e7ee042e29298f34013"))
#     sleep(5)
```

b) geolocation.py

```
from __future__import print_function
import time
import locationiq
from locationiq.restimport ApiException
```

```
from pprintimport pprint
import weatherdata
configuration = locationiq.Configuration()
# Configure API key authorization: key
configuration.api_key['key'] = "pk.a237cd56edda4ec684e2a5bf94f30a71"
# Uncomment below to setup prefix (e.g. Bearer) for API key, if needed
# configuration.api_key_prefix['key'] = 'Bearer'

# Defining host is optional and default to https://eu1.locationiq.com/v1
configuration.host = "https://eu1.locationiq.com/v1"
# Enter a context with an instance of the API client
with locationiq.ApiClient(configuration) as api_client:
    # Create an instance of the API class
    api_instance = locationiq.AutocompleteApi(api_client)
    q = 'Hospital' # str | Address to geocode
    qq = 'College'
normalizecity = 1# int | For responses with no city value in the address section, the next available element
in this order - city_district, locality, town, borough, municipality, village, hamlet, quarter, neighbourhood -
from the address section will be normalized to city. Defaults to 1 for SDKs.
limit = 10# int | Limit the number of returned results. Default is 10. (optional) (default to 10)
viewbox = '12.8769, 80.1762,13.2197, 80.5091' # str | The preferred area to find search results.  To restrict
results to those within the viewbox, use along with the bounded option. Tuple of 4 floats. Any two corner
points of the box - `max_lon,max_lat,min_lon,min_lat` or `min_lon,min_lat,max_lon,max_lat` - are
accepted in any order as long as they span a real box.  (optional)
bounded = 0# int | Restrict the results to only items contained with the viewbox (optional)
countrycodes = 'in'# str | Limit search to a list of countries. (optional)
accept_language = 'en'

# lon = 80.2785
# lat = 13.0878

def geoloc(lat,lon,chkdist):
    a=[]
    print("Starting GeoLocation Service")
    try:
        api_response_hosp = api_instance.autocomplete(q, normalizecity, limit=limit, viewbox=viewbox,
bounded=bounded, countrycodes=countrycodes, accept_language=accept_language)
        api_response_clg = api_instance.autocomplete(qq, normalizecity, limit=limit, viewbox=viewbox,
bounded=bounded, countrycodes=countrycodes, accept_language=accept_language)
        fori inapi_response_hosp:
            x=i['lat']
            y=i['lon']
```

```
            if (abs((float(x)-lat) <= chkdist) & (abs(float(y)-lon) <= chkdist)):
                a.append(i["display_place"])
        fori inapi_response_clg:
            x=i['lat']
            y=i['lon']
            if (abs((float(x)-lat) <= chkdist) & (abs(float(y)-lon) <= chkdist)):
                a.append(i["display_place"])
        # pprint(api_response)
    exceptApiException ase:
        print("Exception when calling AutocompleteApi->autocomplete: %s\n" % e)
    out = ' '.join(map(str,a))
    returnout
# geoloc(13.08,80.2,0.02)
```

c) noderedpush.py

```
import wiotp.sdk.device
import time

myConfig = {
    "identity" : {
        "orgId" : "gsavkf",
        "typeId" : "RaspberryPi",
        "deviceId" : "2019504030"
        },
    "auth" : {
    "token" : "9876543210"
    }
}




def myCommandCallback(cmd):
    print("recieved cmd : ",cmd)


def logData2Cloud(location,temperature,visibility,wind,wind_dir,sl,nohonk,zone):
    client = wiotp.sdk.device.DeviceClient(config=myConfig,logHandlers=None)
    client.connect()
    client.publishEvent(eventId="status",msgFormat="json",data={
        "temperature" : temperature,
        "visibility" : visibility,
```

```
        "location" : location,
        "wind" : wind,
        "wind_dir" : wind_dir,
        "speed_limit" : sl,
        "Status" : nohonk,
        "zone" : zone
    },qos=0,onPublish=None)
    client.commandCallback = myCommandCallback
    client.disconnect()
    time.sleep(1)
```

d) algo.py

```
from datetimeimport datetime

import weatherdata
from geolocationimport geoloc
from noderedpushimport logData2Cloudas log2cloud

# from time import sleep

def mainfunc(myLocation,APIKEY,lat,lon,sl,act_time):
    weatherData = weatherdata.get(myLocation,APIKEY)
    nearby_place = geoloc(lat,lon,0.02)
    # finalSpeed = sl if weatherData["wind"][1] < 25 else sl/1.5
    finalSpeed = sl if"rain" notin weatherDataelse sl/2
    finalSpeed = finalSpeed ifweatherData["visibility"]>3.5else finalSpeed/2

    if("Hospital"in nearby_place):      # hospital zone
        zone="Hospital Zone"
        noHonk = "Do not Honk"
    else:
        if("College"not innearby_place):         # neither school nor hospital zone
            zone="Ride Safely"
            noHonk = "Honk if Needed"
        else:           # school zone
            now = [datetime.now().hour,datetime.now().minute]
            activeTime = [list(map(int,_.split(":"))) for _in act_time]
            noHonk = "Do not Honk"if (activeTime[0][0]<=now[0]<=activeTime[1][0] and
activeTime[0][1]<=now[1]<=activeTime[1][1]) else "Honk if Needed"
            zone="School Zone"
    out={"speed" : finalSpeed,"noHonk" : noHonk}
```

```
log2cloud(myLocation,weatherData["temperature"],weatherData["visibility"],weatherData["wind"],weatherData["wind_dir"],out["speed"],out["noHonk"],zone)
    return(out)
```

e) main.py

```
#Main Microcontroller Code
from timeimport sleep
import algo


loc = "Chennai"
lat = 13.08         #GPS Input Simulation
lon = 80.2         #GPS Input Simulation
APIkey= "c132fedc6afa3e7ee042e29298f34013"
sl = 40
act_time = ["7:30", "17:30"]
while True:
    out=algo.mainfunc(loc,APIkey,lat,lon,sl,act_time)
    print(out)
    sleep(5)
```

Output: