Smart Flood-Level Monitoring and Alert System Using ESP32 and Firebase

Pugalarasu M,
Vellore Institute of Technology,
Vellore, Tami Nadu.

Abstract - Flooding caused by heavy rainfall, absence of drainage, or rising water bodies has been a regular phenomenon in the majority of the urban and semi-urban regions. The floods typically manifest with little or no notice and result in suspended transportation, accidents, loss of property, and risk to human life. The primary problem is the unavailability of low-cost real-time flood observation systems that can be rapidly installed in risk zones.
This paper presents the design and implementation of a Smart Flood-Level Monitoring and Alert System based on IoT (Internet of Things) technology. The system employs an ESP32 microcontroller, which reads the water levels from an ultrasonic sensor, and transmits the readings to the Firebase Realtime Database via Wi-Fi. An Android mobile app is employed as the user interface, and users can monitor road conditions in real time.
It is a costeffective, scaleable, modifiable system which can help people and local administrat ions make informed decisions during flooding conditions.

System Methodology and Workflow

1. Measurement Phase

The measurement phase is the first part of the flood monitoring system. It is triggered by the HC-SR04 ultrasonic sensor, mounted above the road surface in a stationary position. It operates by sending ultrasonic waves at a frequency of 40 kHz.
These are sent through the air, bounce off the surface beneath—pooled water or road itself—and return to the sensor. The duration it takes for this echo back is calculated. From the air speed of sound (approximately 343 m/s), the sensor calculates the distance from itself to the surface of the road. This calculated distance is an instantaneous reading of the present water level. In the dry state, the distance is fixed, varying with initial calibration. When water starts to pool, the calculated distance falls proportionally.
This instantaneous reading is crucial in deciding the presence of floodwater on the road. Accurate measurement is critical, as a miscalculation in a single sensing will lead to false alarms or misclassification of road safety status, and therefore this phase must be highly reliable and monitored in real-time.

2. Processing Phase

At the processing stage, the ESP32 controller reads the raw distance measurements given by the ultrasonic sensor. The system is pre-configured with a baseline distance value representing normal, dry-road conditions. ESP32 continuously compares the live measurement with this baseline. If the live reading senses a significantly shorter distance than the expected one, the controller takes it as a sign of the presence of water between the sensor and road surface.
The level of drop in

distance indicates the amount of water. By comparing the above, ESP32 calculates the road condition status using predefined thresholds. For example, if the distance is more than 25 cm, it's Safe; between 15 and 25 cm, it is Caution; and less than 15 cm, it's Not Safe. The thresholds are user-configurable based on surroundings and road profile. This stage is important not only for live classification but also for filtering minor anomalies that may occur due to wind, trash, or the presence of small animals passing by, which can cause no harm whatsoever.

3. Communication Phase

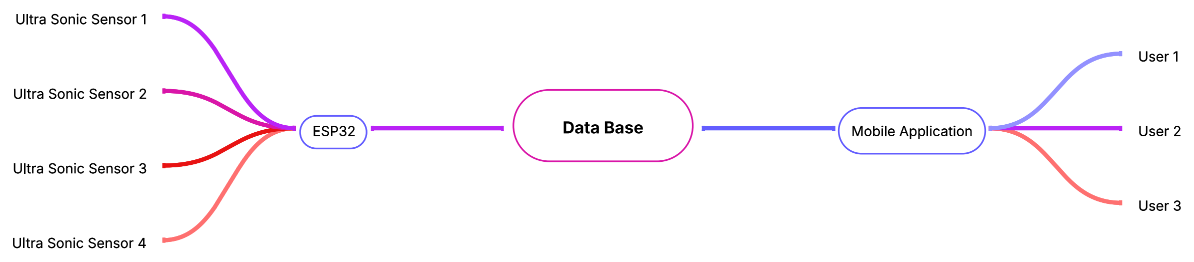
Once the flood level has been evaluated, the system proceeds to the communication stage. In this, the ESP32 sends the processed information—metered distance, calculated flood level status (Safe, Caution, or Not Safe), and reading timestamp—to a cloud-based Firebase Realtime Database. This is performed over Wi-Fi using HTTPS requests to ensure safe and encrypted data transmission. Firebase was employed for its reliability, scalability, and real-time synchronization properties, which are crucial in time-dependent flood alerts. The database is structured in a manner where each ESP32 device corresponds to a single node, such that data retrieval if accessed from a different node is easier. This cloud system is advantageous by having easy access to the data from various points, for instance, mobile apps, web dashboards, or even monitoring equipment in the admin office. Firebase also possesses the capability to trigger real-time updates among devices without time-based polling, conserving latency and power consumption. Success in this stage ensures users and authorities instant notification, and thus it is an essential component of the system's efficiency.

4. Alert Phase

The Android mobile application is the center of user interaction during the alert phase. The application is interfaced with the Firebase Realtime Database and is programmed to fetch new data the instant it is updated by the ESP32 microcontroller. Following data fetch, the application processes the data and presents it in a graphical user-friendly form. Color-coded visual signals—green for safe, yellow for caution, and red for danger—are used to enable quick communication of the flood status, minimizing the amount of interpretation required by users. Apart from the color signals, the application can also present the actual water level (in centimeters), the last update time, and the precise road location being monitored. This renders it highly convenient for users to make real-time route decisions. In future releases, the application can also give voice warnings or push notifications, especially during emergencies. Responsiveness and intelligibility of this alert phase are key to user confidence and large-scale implementation of the system.

5. Scalability Consideration

Scalability is a primary design consideration, particularly for real-world deployment across cities or regions. The system design is inherently modular, and multiple ESP32 devices can potentially be executed individually and in parallel. Each of the sensor nodes has a unique dedicated identifier and posts its readings to a dedicated path in the Firebase Realtime Database. This prevents data from different points from interfering with each other or getting in each other's way, and monitoring of all the flood-vulnerable points is simplified. This design facilitates the simple addition of large numbers of sensor nodes without involving heavy code updates or additional hardware. Additionally, administrative users or city authorities are provided with a common dashboard to view combined representations of flood conditions across many different points of location in real time. Integration into the cloud allows the possibility of scaling the geographic footprint of the system infinitely, subject to network connectivity and device power sources. Also, future integration with low-power wide-area network (LPWAN) technology such as LoRa or GSM allows the system's deployment in low or no infrastructure sites. Scalability makes the system not only an effective solution to deploy and run, but as part of an extended smart city or disaster response system too.



Program :

Mobile Application Code :

Flutter :

```
//main Page
import
'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:myapp/firebase_options.dart';
import 'package:myapp/home_page.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(options:
DefaultFirebaseOptions.currentPlatform);
  runApp(const MyApp());
}

class MyApp extends StatefulWidget {
  const MyApp({super.key});

  @override
  State<MyApp> createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      debugShowCheckedModeBanner: false,
      home: HomePage(),
    );
  }
}
//map Page
import
'package:firebase_core/firebase_core.dart';
import
'package:firebase_database/firebase_database.d
art';
import 'package:flutter/material.dart';
import
'package:google_maps_flutter/google_maps_flu
tter.dart';

class HomePage extends StatefulWidget {
  const HomePage({super.key});

  @override
  State<HomePage> createState() =>
_HomePageState();
}
```

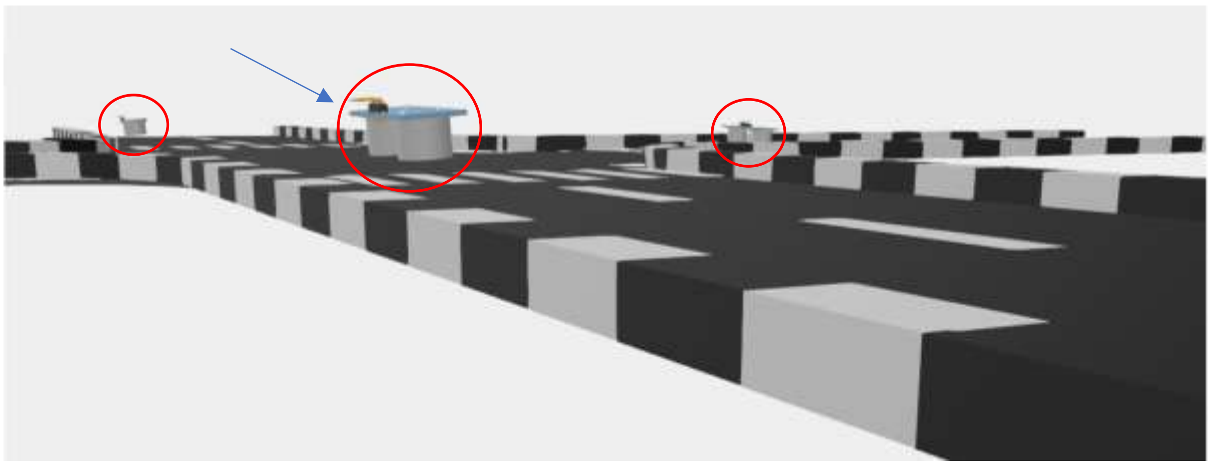
```
class _HomePageState extends
State<HomePage> {
  final Future<FirebaseApp> _fApp =
Firebase.initializeApp();

  // Global key to manage Snackbar
  final GlobalKey<ScaffoldMessengerState>
_scaffoldKey =
GlobalKey<ScaffoldMessengerState>();

  String realTimeValue = "0";
  String realTimeValue2 = "0";
  String realTimeValue3 = "0";
  String realTimeValue4 = "0";

  @override
  Widget build(BuildContext context) {
    return SafeArea(
      child: ScaffoldMessenger(
        key: _scaffoldKey,
        child: Scaffold(
          extendBodyBehindAppBar: true,
          appBar: AppBar(
            toolbarHeight: 100,
            backgroundColor: Colors.transparent,
            elevation: 0,
            centerTitle: true,
            title: Container(
              height: 70,
              width: double.infinity,
  padding: const EdgeInsets.symmetric(vertical:
20, horizontal: 30),
  decoration: BoxDecoration(
    gradient: const LinearGradient(
      colors: [Color.fromARGB(255, 189, 222, 249),
Color.fromARGB(255, 131, 255, 162)],
      begin: Alignment.topLeft,
      end: Alignment.bottomRight,
    ),
    borderRadius: BorderRadius.circular(25),
    boxShadow: [
      BoxShadow(
        color: Colors.black.withOpacity(0.3),
        blurRadius: 10,
        offset: const Offset(0, 4),
      ),
    ],
  ),
  child:  Column(
    mainAxisSize: MainAxisSize.min,
    children: [
      ShaderMask(
  shaderCallback: (bounds) => const
LinearGradient(
    colors: [Color.fromARGB(179, 120, 99, 255),
Color.fromARGB(255, 164, 99, 255)],
    begin: Alignment.topLeft,
    end: Alignment.bottomRight,
  ).createShader(bounds),
  child: const Text(
    "Water Level Monitoring",
    style: TextStyle(
      fontSize: 24,                // Increased font size
for visibility
      fontWeight: FontWeight.w900,   // Extra bold
for strong appearance
      letterSpacing: 1.2,          // Slight spacing for
elegance
      color: Colors.white,         // Base color for the
gradient
    ),
  ),
)


    ],
  ),
)
,
          ),
          body: FutureBuilder(
            future: _fApp,
            builder: (context, snapshot) {
              if (snapshot.hasError) {
                return const Center(child:
Text("Something went wrong"));
              } else if (snapshot.hasData) {
                return content();
              } else {
                return const Center(child:
CircularProgressIndicator());
              }
            },
          ),
        ),
      ),
    );
  }

  Widget content() {
    DatabaseReference testRef =
      FirebaseDatabase.instance.ref("sensors").ch
ild("sensor1");
   DatabaseReference testRef2 =
      FirebaseDatabase.instance.ref("sensors").ch
ild("sensor2");
   DatabaseReference testRef3 =
      FirebaseDatabase.instance.ref("sensors").ch
ild("sensor3");
   DatabaseReference testRef4 =
      FirebaseDatabase.instance.ref("sensors").ch
ild("sensor4");

   testRef.onValue.listen((event) {
     setState(() {
      realTimeValue =
event.snapshot.value.toString();
     });
   });

   testRef2.onValue.listen((event) {
     setState(() {
      realTimeValue2 =
event.snapshot.value.toString();
     });
   });

   testRef3.onValue.listen((event) {
     setState(() {
      realTimeValue3 =
event.snapshot.value.toString();
     });
   });

   testRef4.onValue.listen((event) {
     setState(() {
      realTimeValue4 =
event.snapshot.value.toString();
     });
   });

   return Scaffold(
     body: GoogleMap(
      rotateGesturesEnabled: false,
      zoomControlsEnabled: false,
      zoomGesturesEnabled: true,
      minMaxZoomPreference: const
MinMaxZoomPreference(18.9, 19.1),
      cameraTargetBounds: CameraTargetBounds(
        LatLngBounds(
          southwest: const
LatLng(12.968981241763402,
79.15715192358472),
          northeast: const
LatLng(12.971197854350912,
79.15930083404112),
        ),
      ),
      polylines: {
        // Road 1
        Polyline(
          consumeTapEvents: true,
          onTap: () => showPopup("Road 1",
realTimeValue),
          width: 18,
          endCap: Cap.roundCap,
          startCap: Cap.roundCap,
          polylineId: const PolylineId("road1"),
          color: getColor(realTimeValue),
          points: const [
            LatLng(12.969791878611076,
79.15814448655277),
            LatLng(12.97017864280351,
79.15816382249426),
            LatLng(12.970600758465286,
79.15818949167353),
            LatLng(12.970925943389808,
79.15831142027925),
            LatLng(12.97098782602536,
79.15833056198899),
            LatLng(12.971884571378316,
79.15821327807767),
          ],
        ),

        // Road 2
        Polyline(
          consumeTapEvents: true,
          onTap: () => showPopup("Road 2",
realTimeValue2),
          width: 18,
          endCap: Cap.roundCap,
          startCap: Cap.roundCap,
          polylineId: const PolylineId("road2"),
          color: getColor(realTimeValue2),
          points: const [
            LatLng(12.969721392161171,
79.15805496652217),
            LatLng(12.96973054040014,
79.15764056526025),
          LatLng(12.969665195827178,
79.15746085724197),
          LatLng(12.969659968259792,
79.15699817620664),
          LatLng(12.96968218541645,
79.15668569883974),
          LatLng(12.969671730283906,
79.15643625341136),
         ],
        ),

        // Road 3
        Polyline(
         consumeTapEvents: true,
         onTap: () => showPopup("Road 3",
realTimeValue3),
         width: 18,
         endCap: Cap.roundCap,
         startCap: Cap.roundCap,
         polylineId: const PolylineId("road3"),
         color: getColor(realTimeValue3),
         points: const [
          LatLng(12.969649093681284,
79.1580956191877),
          LatLng(12.969548463001146,
79.15808891366586),
          LatLng(12.969477890811573,
79.15809964250082),
          LatLng(12.96942953578123,
79.15812780569256),
          LatLng(12.9693171429717,
79.1582055897459),
          LatLng(12.969206417850014,
79.15825216753169),
          LatLng(12.969141366885665,
79.15817295758741),
          LatLng(12.96907340837006,
79.15810858457775),
         ],
        ),

        // Road 4
        Polyline(
         consumeTapEvents: true,
         onTap: () => showPopup("Road 4",
realTimeValue4),
         width: 18,
         endCap: Cap.roundCap,
         startCap: Cap.roundCap,
         polylineId: const PolylineId("road4"),
         color: getColor(realTimeValue4),
         points: const [
          LatLng(12.969711824476347,
79.15822838854446),
          LatLng(12.96971051758499,
79.15834372352013),
          LatLng(12.969693527996684,
79.15863206095933),
          LatLng(12.969696141779586,
79.15866156525544),
          LatLng(12.970999109152693,
79.15869911619676),
          LatLng(12.969696141779586,
79.15866156525544),
          LatLng(12.968720546550369,
79.15864770856258),
         ],
        )
       },
       initialCameraPosition: const
CameraPosition(
        zoom: 18,
        target: LatLng(12.96980695531399,
79.15810869611384),
       ),
      ),
    );
  }

  // Custom snackbar with card-like UI
  void showPopup(String road, String value) {
   _scaffoldKey.currentState?.hideCurrentSnackB
ar();
   _scaffoldKey.currentState?.showSnackBar(
    SnackBar(
     duration: const Duration(seconds: 3),
     behavior: SnackBarBehavior.floating,
     backgroundColor: Colors.transparent,
     elevation: 0,
     content: Card(
      shape:
RoundedRectangleBorder(borderRadius:
BorderRadius.circular(12)),
      elevation: 6,
      color: getColor(value).withOpacity(0.9),
      child: ListTile(
       leading: const Icon(Icons.water_drop,
color: Colors.white, size: 30),
       title: Text(
        "$road: Water Level = $value cm",
        style: const TextStyle(color: Colors.white,
fontWeight: FontWeight.bold),
       ),
        ),
      ),
    ),
  );
 }

 Color getColor(String value) {
   double level = double.tryParse(value) ?? 0.0;
   if (level > 30) return Colors.green;
   if (level > 15) return Colors.orange;
   return Colors.red;
 }
}
```

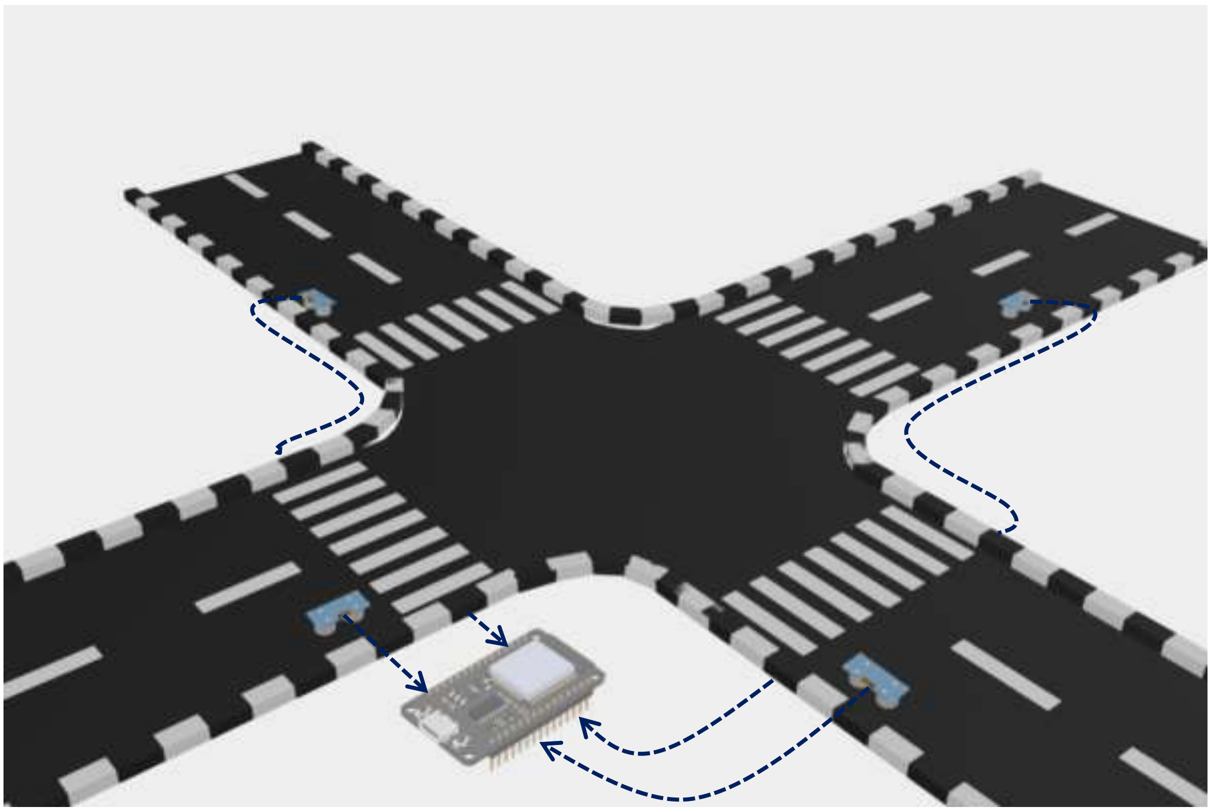
ESP Code :

```
#include <Arduino.h>
#if defined(ESP32)
  #include <WiFi.h>
#elif defined(ESP8266)
  #include <ESP8266WiFi.h>
#endif
#include <Firebase_ESP_Client.h>

// Define pins for the four sensors
const int trigPins[] = {15,4,18,22}; // Trig pins for
sensors
const int echoPins[] = {2,5,19,23}; // Echo pins for
sensors

// Define sound velocity in cm/µs
#define SOUND_VELOCITY 0.034

// Variables for sensor readings
long durations[4];
float distancesCm[4];

// Firebase includes
#include "addons/TokenHelper.h"
#include "addons/RTDBHelper.h"

// Insert your network credentials
#define WIFI_SSID "vivo Y16"
#define WIFI_PASSWORD "12345678"

// Insert Firebase project API Key
#define API_KEY
"AIzaSyCVJZ8pLpzaQTul0N6DotQAIrPwsUTbI7s"

// Insert Firebase Realtime Database URL
#define DATABASE_URL "https://esp-fed66-default-
rtdb.asia-southeast1.firebasedatabase.app/"

// Define Firebase Data object
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;
bool signupOK = false;

void setup() {
  Serial.begin(115200); // Start the serial
communication

  // Set up pins for all sensors
  for (int i = 0; i < 4; i++) {
    pinMode(trigPins[i], OUTPUT);
    pinMode(echoPins[i], INPUT);
  }

  // Connect to Wi-Fi
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(300);
  }
  Serial.println();
  Serial.print("Connected with IP: ");
  Serial.println(WiFi.localIP());
  Serial.println();

  // Configure Firebase
  config.api_key = API_KEY;
  config.database_url = DATABASE_URL;

  if (Firebase.signUp(&config, &auth, "", "")) {
    Serial.println("Firebase sign-up successful");
    signupOK = true;
  } else {
    Serial.printf("%s\n",
config.signer.signupError.message.c_str());
  }

  config.token_status_callback = tokenStatusCallback;
// Callback for token generation
  Firebase.begin(&config, &auth);
  Firebase.reconnectWiFi(true);
}

void loop() {
  if (Firebase.ready() && signupOK) {
    for (int i = 0; i < 4; i++) {
      // Trigger and measure distance for each sensor
      digitalWrite(trigPins[i], LOW);
      delayMicroseconds(2);
      digitalWrite(trigPins[i], HIGH);
      delayMicroseconds(10);
      digitalWrite(trigPins[i], LOW);
      durations[i] = pulseIn(echoPins[i], HIGH);
      distancesCm[i] = durations[i] * SOUND_VELOCITY
/ 2;

      // Print sensor readings to the Serial Monitor
      Serial.print("Sensor ");
    Serial.print(i + 1);
    Serial.print(" Distance (cm): ");
    Serial.println(distancesCm[i]);

    // Update Firebase Database with sensor readings
    String path = "Nodemcu 1/Sensor" + String(i + 1);
    Firebase.RTDB.setFloat(&fbdo, path.c_str(),
distancesCm[i]);
   }
  }
  delay(1000); // Delay before the next reading
}
```

Hardware and Simulation Output Results

Real-time Simulation :





Hardware Used

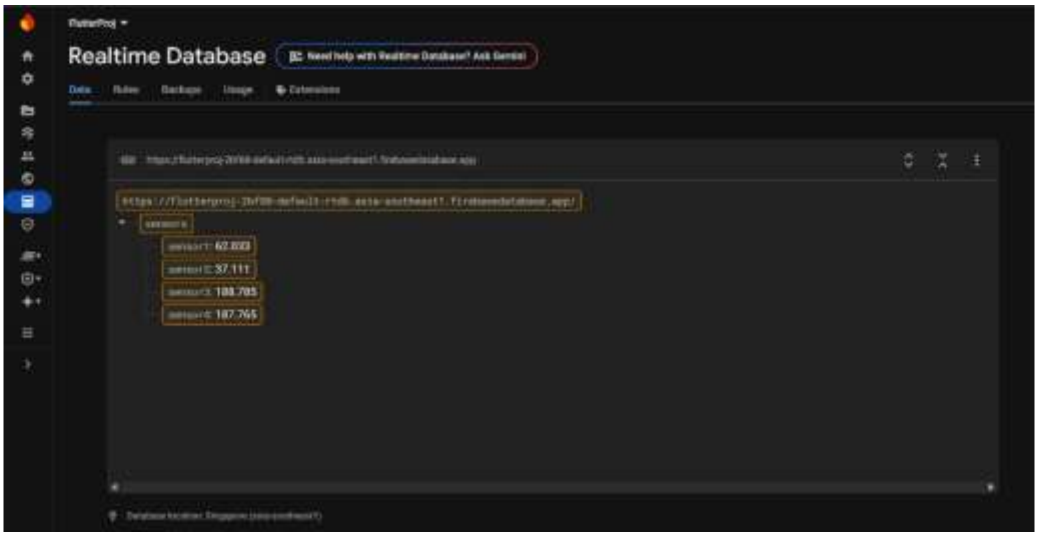
- **ESP32**: Serves as the IoT controller with integrated Wi-Fi.
- **HC-SR04 Ultrasonic Sensor**: Measures distance from the sensor to the road surface.
- **Power Supply**: USB-powered for prototype; solar panels or batteries for real-world deployment.
- **Android Smartphone**: To run the companion mobile application.

Expected Output

- The sensor measures a dry road distance of approximately 30–35 cm.
- As water rises, the distance reduces.
- Status classification:
  - **> 25 cm** – Safe
  - **15–25 cm** – Caution
  - **< 15 cm** – Not Safe
- Real-time updates are reflected in the app with less than 3 seconds delay.

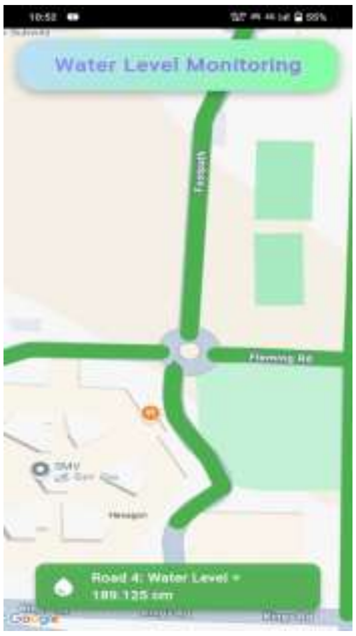
Results :

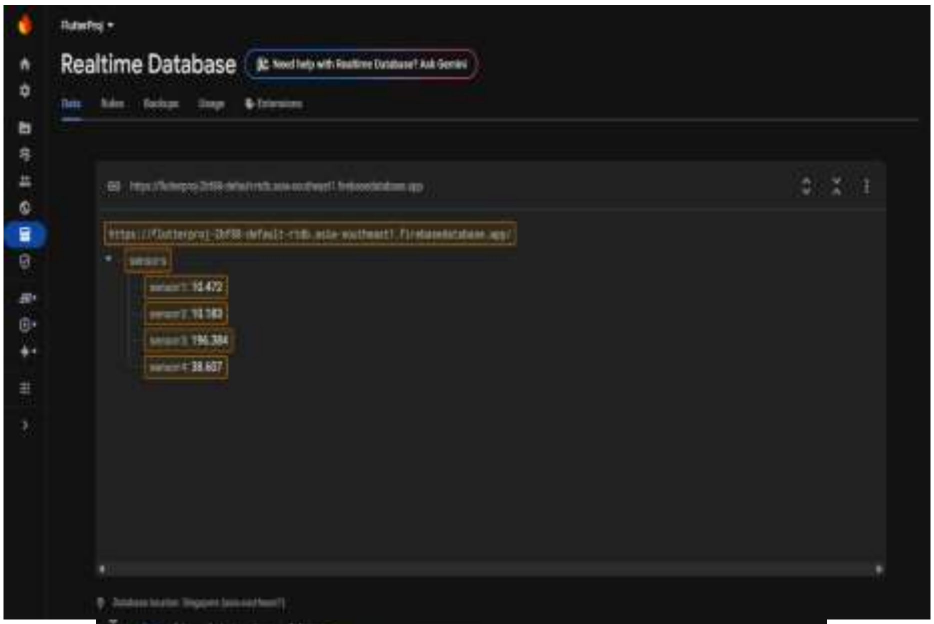
- FireBase as Database,gets the data from esp32 and stores.



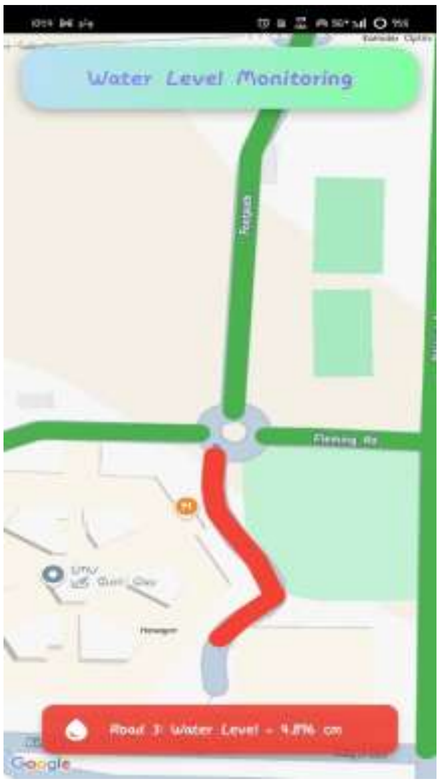
```
https://flutterproj-2bf08-default-rtdb.asia-southeast1.firebasedatabase.app/
▾ sensors
    sensor1: 62.033
    sensor2: 37.111
    sensor3: 188.705
    sensor4: 187.765
```

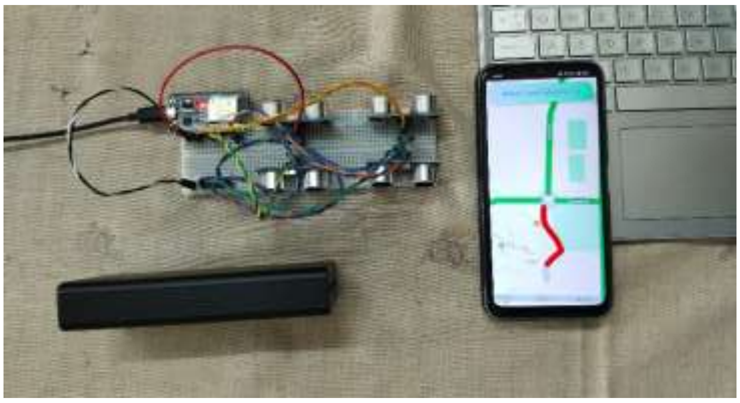
- No Road with water



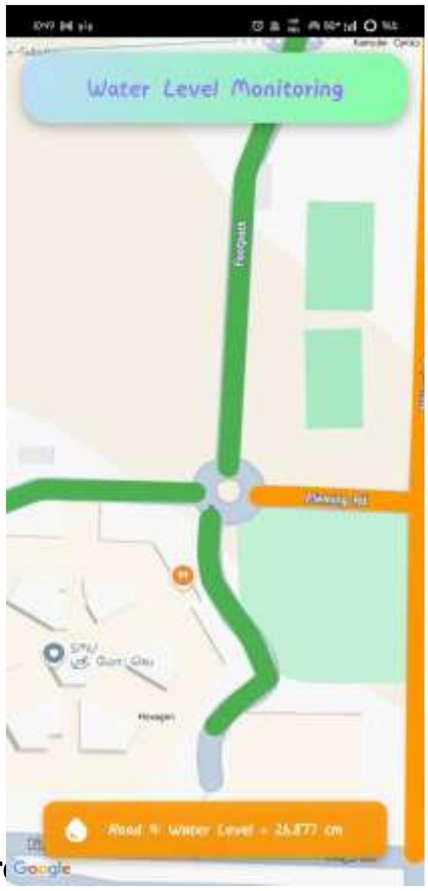


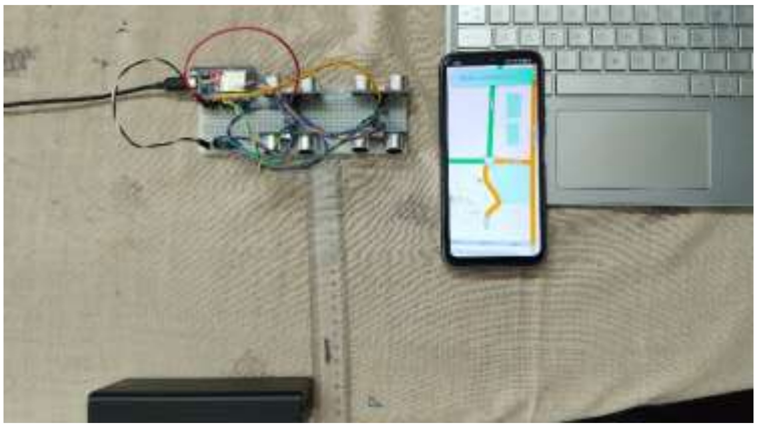
```
https://flutterproj-2bf08-default-rtdb.asia-southeast1.firebasedatabase.app/
▾ sensors
    sensor1: 10.472
    sensor2: 10.183
    sensor3: 196.384
    sensor4: 38.607
```



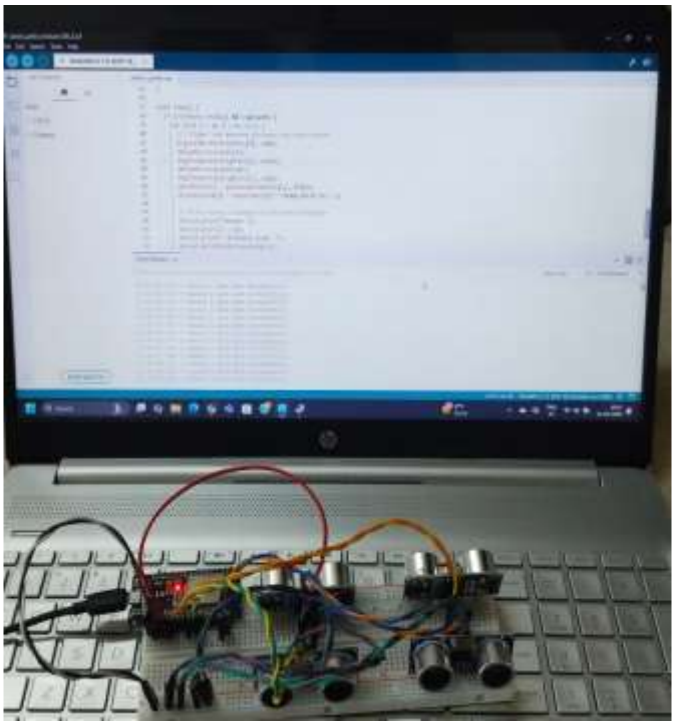
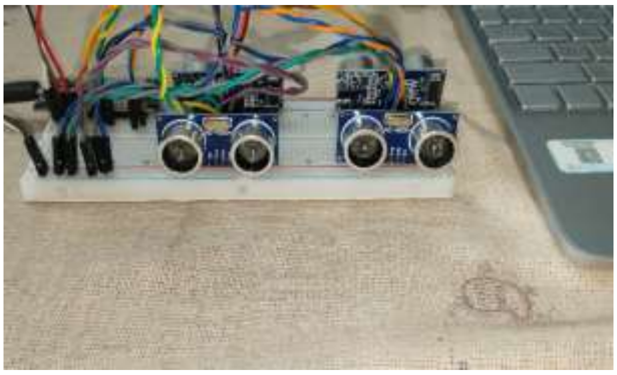
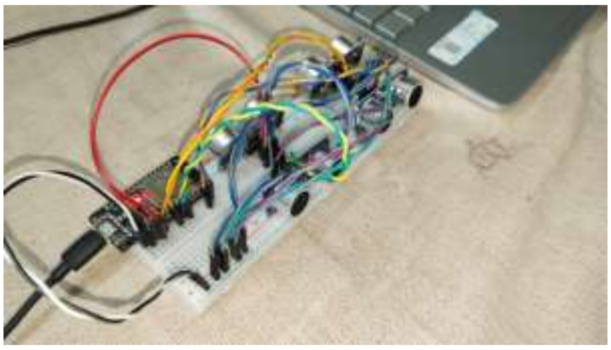


- Road with Moderate level of water





Pr

Theoretical Background

IoT in Disaster Management

IoT systems enable real-time data collection and transmission, ideal for emergency management. In this system, ESP32 acts as an edge device collecting environmental data and uploading it to the cloud for instant access.

Working Principle of Ultrasonic Sensor

The HC-SR04 operates by emitting ultrasonic pulses at 40 kHz. The time it takes for the echo to return is used to calculate the distance using the formula:

Distance = (Time × Speed of Sound) / 2

Why Firebase?

Firebase is a Google-backed cloud platform offering:

- **Realtime Database**: Enables live sync between device and application.
- **Scalability**: Easily supports multiple sensor nodes.
- **Security Rules**: Allows access control and data validation.

Applications and Use Cases

- **Urban Roads**: Automatically notify commuters of waterlogged routes.
- **Highways and Bridges**: Detect dangerous conditions and reroute traffic.
- **Institutional Campuses**: Warn students/staff about unsafe walkways or parking areas.
- **Smart Cities**: Integrate with traffic lights and signage for automated alerts.
- **Emergency Services**: Assist authorities in flood response planning.

Advantages

- **Low-Cost**: Utilizes inexpensive components.
- **Modular**: Easy to add more sensors or modify existing setup.
- **Remote Monitoring**: Data accessible from anywhere via the internet.

- **Real-Time Feedback**: Immediate awareness of ground-level flooding.
- **Eco-Friendly**: Can be solar powered for energy efficiency.

Limitations and Future Enhancements

Current Limitations

- Relies on Wi-Fi availability.
- May produce false readings in extreme weather conditions (fog, wind).
- Limited platform availability (only Android app currently).

Planned Enhancements

- **GPS Integration** for location-based alerts.
- **Push Notifications** to warn users of flooding on their regular routes.
- **Web Dashboard** for centralized monitoring by civic authorities.
- **LoRa/GSM Integration** for remote, low-bandwidth regions.
- **Machine Learning** to predict floods based on weather and historical data.

Conclusion

This project demonstrates a viable smart flood-monitoring system using low-cost IoT hardware. With real-time cloud integration and an intuitive mobile interface, this solution empowers individuals and governments to respond proactively to flood risks. It exemplifies how embedded systems, cloud computing, and mobile technology can work in harmony to address real-world safety challenges.

References

[1] Google, "Firebase," [Online]. Available: https://firebase.google.com/.

[2] O. S. A. a. M. S. O. O. E. Oduwole1, " A MOBILE APPLICATION FOR ROUTE OPTIMIZATION," *SHORTEST ROUTE: A MOBILE APPLICATION FOR ROUTE OPTIMIZATION USING DIGITAL MAP,* pp. 10-11, 2018-2019.

[3] I. A. L. Vito Lauda Putra Anta1, "Disaster Management," *Mobile Application for flood disaster in Jakarta,* pp. 5-6, 2021.

[4] "CircuitDigest IoT Projects," [Online]. Available: https://circuitdigest.com.

Mobile App Link(Android only) – *click here*