

Internet Of ThingS

Internet Of ThingsPublic transport optimization using IOT sensorsPhase



Phase 4:Development Part 2

# \*\*Abstract:\*\*

The Internet of Things (IoT) has catalyzed innovations across various domains, and its application in public transportation systems has demonstrated significant potential for efficiency enhancement. This project, designated as Phase 4 of our ongoing effort, continues the endeavor to harness IoT sensors to optimize public transport operations.

Building upon the foundations of the preceding phases, Phase 4 extends the network of IoT sensors deployed in buses, bus stops, and key transit points to collect real-time data encompassing passenger counts, vehicle conditions, and traffic parameters. The sensor data feeds into a sophisticated IoT platform, which enables comprehensive analysis, predictive maintenance, and route optimization.

### The core objectives of this phase include:

1. \*\*Enhanced Passenger Experience:\*\* The IoT sensors provide real-time passenger information, reducing waiting times and improving service reliability. Passengers can access accurate bus arrival times and occupancy status through mobile apps and public displays.

2. \*\*Operational Efficiency:\*\* The collected data facilitates predictive maintenance, reducing bus downtime and operational disruptions. It also enables dynamic route adjustments based on traffic conditions, further enhancing on-time performance.

3. \*\*Environmental Impact:\*\* By optimizing routes and reducing idling times, the project contributes to reduced fuel consumption and greenhouse gas emissions, promoting a more sustainable urban transport ecosystem.

4. \*\*Data-Driven Decision-Making:\*\* The data analytics component of the IoT platform provides invaluable insights into transportation patterns, enabling authorities to make informed decisions regarding fleet expansion, service adjustments, and infrastructure improvements.

5. \*\*Scalability and Cost Efficiency:\*\* Phase 4 continues to explore cost-effective sensor technologies, striving for scalability and affordability in the deployment of IoT sensors across a wider network of public transport assets.

This project highlights the transformative impact of IoT on public transportation systems, showing promise in improving the passenger experience, reducing operational costs, and contributing to more sustainable urban environments. The lessons and outcomes from Phase 4 inform the ongoing evolution of IoT-driven public transport optimization and serve as a blueprint for cities and transportation authorities looking to modernize their systems.

In summary, Phase 4 of the "Public Transport Optimization Using IoT Sensors" project represents a critical step forward in reimagining urban transportation through data-driven insights, enhanced passenger services, and sustainable practices.

---

This abstract provides a high-level overview of the project, its objectives, and the potential benefits of applying IoT sensors to optimize public transport. You can adapt this abstract to your specific project by including relevant details and goals for your Phase 4 initiative.

# GPS BASSED VIHICLE TRACKING SYSTEM;

### - Real-Time Location Awareness: Empowering IoT for a Connected World;

Real-time location awareness, the ability to pinpoint the precise geographic coordinates and status of objects, vehicles, or individuals at any given moment, has emerged as a pivotal facet of the Internet of Things (IoT) ecosystem. This abstract explores the significance and applications of real-time location awareness in our increasingly connected world.

In today's data-driven landscape, the demand for real-time location information is escalating across various industries. From transportation and logistics to healthcare, public safety, and asset management, the need for accurate, up-to-the-second location data has never been more critical. This project delves into the technologies and methodologies that underpin real-time location awareness.



Key components and themes of this exploration include:

1. \*\*Geospatial Technology Integration:\*\* Real-time location awareness relies on the integration of GPS, GNSS (Global Navigation Satellite System), and IoT sensors. This synergy provides a wealth of data that enables the accurate determination of an entity's location and movement.

2. \*\*IoT Platform and Data Analytics:\*\* The collected location data is transmitted to centralized IoT platforms, where it undergoes real-time analysis. The data analytics component offers valuable insights into routes, patterns, and trends, empowering informed decision-making.

3. \*\*Geofencing and Alerts:\*\* Geofencing features allow for the creation of virtual boundaries,

for security, safety, and asset protection.

4. \*\*Operational Efficiency:\*\* Industries such as logistics and transportation benefit from real-time location awareness by optimizing routes, ensuring on-time deliveries, and enhancing resource allocation.

5. \*\*Personal Safety and Healthcare:\*\* In healthcare and personal safety applications, real-time location awareness enables tracking of individuals, including vulnerable populations and lone workers, enhancing their safety and well-being.

6. \*\*Privacy and Ethical Considerations:\*\* The project acknowledges the importance of addressing privacy concerns and ethical considerations related to tracking and surveillance. It explores mechanisms for responsible and secure data handling.

7. \*\*Scalability and Future Prospects:\*\* The project emphasizes the scalability and future potential of real-time location awareness, paving the way for new applications and services that can further connect and streamline the world.

In conclusion, the project explores the transformative role of real-time location awareness, highlighting its impact on enhancing operational efficiency, safety, and connectivity. This technology stands at the intersection of IoT, geospatial data, and data analytics, offering myriad possibilities for a more connected and informed world.

---

You can tailor this abstract to suit the specific focus and goals of your project or research related to real-time location awareness.

# Integration of Iot devices:

Connect the VCC pin of the GPS module to the ESP32 3.3V pin.

1. Connect the GPS ground pin to the ESP32 ground pin.

2. Connect the RX pin of the GPS module to the TX pin of the ESP32.

3. Connect your ESP32 to the computer through a USB cable.

4. Program the ESP32 via Arduino IDE with python to calculate latitude and

longitude information from gps data .

5. Integrate this setup to public vehicle.

6. Send the GPS data, read by the ESP32 from the GPS device, to an external

web server to store it on the real time database(Firebase

# ULTRASONIC SENSOR:

Integrating an ultrasonic sensor with an Arduino Uno is a common and straightforward project. An ultrasonic sensor is used to measure distance by sending out a sound wave and measuring the time it takes for the wave to bounce back. Here's a step-by-step guide on how to do it:

\*\*Components Needed:\*\*

1. Arduino Uno

2. Ultrasonic sensor (commonly used HC-SR04)

3. Jumper wires

4. Breadboard (optional)

\*\*Wiring:\*\*

1. Connect the VCC pin of the ultrasonic sensor to the 5V pin on the Arduino Uno.

2. Connect the GND (Ground) pin of the ultrasonic sensor to the GND (Ground) pin on the Arduino Uno.

3. Connect the Trig (Trigger) pin of the ultrasonic sensor to a digital pin on the Arduino Uno (e.g., Pin 7).

4. Connect the Echo pin of the ultrasonic sensor to another digital pin on the Arduino Uno (e.g., Pin 6).

Here's a simple Arduino code to read data from the ultrasonic sensor and print the distance in centimeters:

```arduino

// Define the pins for the ultrasonic sensor

const int trigPin = 7;

const int echoPin = 6;

void setup() {

  // Initialize serial communication

  Serial.begin(9600);

  // Set the trigPin as an OUTPUT

  pinMode(trigPin, OUTPUT);

  // Set the echoPin as an INPUT

  pinMode(echoPin, INPUT);

}

void loop() {

  // Clear the trigPin

  digitalWrite(trigPin, LOW);

  delayMicroseconds(2);

  // Set the trigPin on for 10 microseconds

  digitalWrite(trigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(trigPin, LOW);

  // Read the echoPin, and calculate the duration in microseconds

  long duration = pulseIn(echoPin, HIGH);

  // Calculate the distance in centimeters

  // Speed of sound is approximately 343 meters per second or 0.0343 cm per microsecond

  int distanceCM = duration \* 0.0343 / 2;

  // Print the distance to the Serial Monitor

  Serial.print("Distance: ");

  Serial.print(distanceCM);

  Serial.println(" cm");

  // Delay before taking another reading

  delay(1000);

}

```

\*\*Upload the code to your Arduino Uno\*\*:

1. Open the Arduino IDE on your computer.

2. Connect your Arduino Uno to your computer via USB.

3. Select the correct board and port under the "Tools" menu.

4. Copy and paste the above code into the Arduino IDE.

5. Click the "Upload" button to upload the code to your Arduino Uno.

Once the code is uploaded, open the Serial Monitor (Tools -> Serial Monitor) to see the distance measurements in centimeters. The sensor will continuously measure and display the distance. Place an object in front of the sensor to see how it reacts to changes in distance.

Remember to power off the Arduino before making any wiring changes, and be cautious when working with electrical components.

Creating a complete submarine simulation with an ultrasonic sensor (emulating SONAR) in WOKWI using the ESP32 platform is a complex task that may require a considerable amount of time and effort, including 3D modeling and animation. However, I can provide you with a simplified example that demonstrates how you can use an ultrasonic sensor to measure distances in a basic underwater environment in WOKWI. This will serve as a starting point for your project.

Here's a step-by-step guide:

\*\*Step 1: Set up the Circuit\*\*

1. Go to the WOKWI website (https://wokwi.com/).

2. Create a new project and select the ESP32 as your target board.

3. Add the following components to your project:

   - ESP32 (NodeMCU-32S)

   - HC-SR04 ultrasonic sensor

   - LED (for indicating SONAR results, optional)

   - Breadboard and jumper wires

4. Connect the components as follows:

   - Connect VCC of the HC-SR04 to 5V on the ESP32.

   - Connect GND of the HC-SR04 to GND on the ESP32.

   - Connect TRIG of the HC-SR04 to GPIO 12 on the ESP32.

   - Connect ECHO of the HC-SR04 to GPIO 14 on the ESP32.

   - Optionally, connect an LED with a current-limiting resistor to a digital pin on the ESP32 (e.g., GPIO 2).

\*\*Step 2: Create the Code\*\*

Here's an example Arduino code for your ESP32 that simulates the SONAR operation by measuring distances using the ultrasonic sensor and blinking an LED based on the proximity:

```cpp

#include <Arduino.h>

const int trigPin = 12;  // Trigger pin of HC-SR04

const int echoPin = 14;  // Echo pin of HC-SR04

const int ledPin = 2;    // Digital pin for the LED (optional)

void setup() {

  Serial.begin(115200);

  pinMode(trigPin, OUTPUT);

  pinMode(echoPin, INPUT);

  pinMode(ledPin, OUTPUT);

}

void loop() {

  long duration, distance;

  // Clear the trigger pin

  digitalWrite(trigPin, LOW);

  delayMicroseconds(2);

  // Send a 10μs pulse to trigger pin

  digitalWrite(trigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(trigPin, LOW);

  // Read the pulse from echo pin

  duration = pulseIn(echoPin, HIGH);

  // Calculate distance in cm

  distance = (duration / 2) / 29.1;

  // Print the distance to the Serial Monitor

  Serial.print("Distance: ");

  Serial.print(distance);

  Serial.println(" cm");

  // Determine SONAR result and control the LED

  if (distance <= 20) {

    digitalWrite(ledPin, HIGH); // Turn on LED when an object is detected within 20 cm

} else {

    digitalWrite(ledPin, LOW);  // Turn off LED when no object is detected or it's farther away

  }

  delay(1000); // Wait for 1 second before taking another reading

}

```

\*\*Step 3: Run the Simulation\*\*

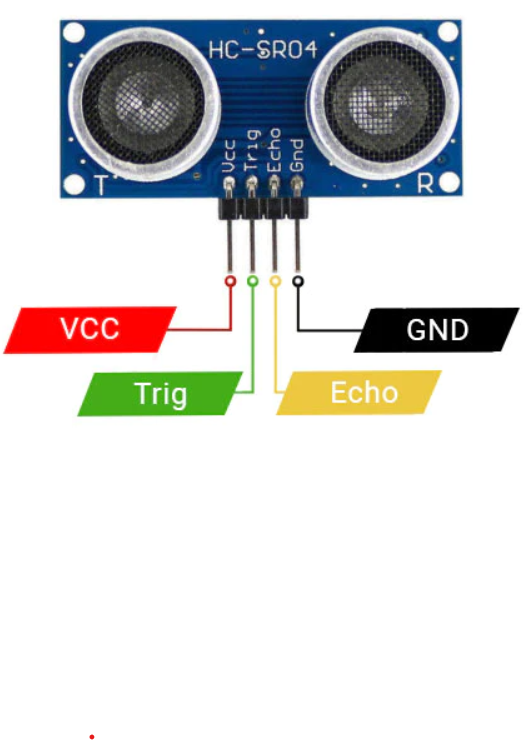
- Upload the code to your ESP32 in WOKWI.

- Run the simulation.

- You should see distance measurements in centimeters in the Serial Monitor.

- If an object is detected within 20 cm, the LED should turn on to simulate SONAR detection.

This example provides a basic starting point for simulating a SONAR-like system using an ultrasonic sensor and an ESP32 on WOKWI. You can further expand and customize the simulation as needed for your submarine project. Please note that creating a full-fledged submarine simulation with realistic 3D modeling and animations would require more advanced tools and development efforts.



GPS data on an OLED display :

\*\*Components You'll Need:\*\*

1. GPS Module or GPS Receiver: This will provide latitude and longitude data.

2. Microcontroller: Such as Arduino, Raspberry Pi, or a microcontroller with GPIO pins.

3. OLED Display: Make sure it's compatible with your microcontroller.

\*\*Step 1: Set Up the Hardware\*\*

1. Connect the GPS module to your microcontroller. Typically, this involves connecting the GPS module's TX and RX pins to the microcontroller's serial pins, and the VCC and GND pins to the appropriate power and ground pins on the microcontroller.

2. Connect the OLED display to your microcontroller. Make sure to connect the appropriate pins for power, ground, data, and clock.

\*\*Step 2: Write the Code\*\*

You'll need to write code to read data from the GPS module and display it on the OLED screen. Below is a simplified example using Arduino and Adafruit's OLED library. You may need to modify it to fit your specific hardware and requirements:

```cpp

#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

#include <SoftwareSerial.h>

SoftwareSerial gpsSerial(4, 3); // RX, TX pins for GPS module

Adafruit\_SSD1306 display(128, 64);

void setup() {

gpsSerial.begin(9600); // GPS module baud rate

display.begin(SSD1306\_I2C\_ADDRESS, OLED\_RESET); // Initialize the OLED display

display.display(); // Display initialization

display.clearDisplay();

}

void loop() {

if (gpsSerial.available() > 0) {

String gpsData = gpsSerial.readStringUntil('\n'); // Read a line of data from GPS module

if (gpsData.startsWith("$GPRMC")) {

// Parse GPS data (NMEA format) and extract latitude and longitude

// Example: $GPRMC,123519,A,4807.038,N,01131.000,E,...

// Extract and parse latitude and longitude here

// Display latitude and longitude on the OLED display

display.clearDisplay();

display.setTextSize(1);

display.setTextColor(SSD1306\_WHITE);

display.setCursor(0,0);

display.println("Latitude: xx.xxxxxx");

display.setCursor(0,12);

display.println("Longitude: xx.xxxxxx");

display.display();

}

}

}

```

Please note that the code above is a basic example and may need modifications depending on the GPS module and OLED display you are using. You'll also need to parse the NMEA data to extract latitude and longitude correctly. Additionally, you may want to display other information like speed, altitude, or a map.

Make sure to install the required libraries for your microcontroller and OLED display and upload the code to your microcontroller.

\*\*Step 3: Test and Debug\*\*

Test your setup and code to ensure it's working correctly. Make sure the GPS module can acquire a GPS fix, and the data is displayed on the OLED screen as expected.

Remember that GPS modules often require a clear line of sight to the sky to get a GPS fix, so testing outdoors or near a window may be necessary for accurate data.#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

##include <TinyGPS++.h>

#include <HardwareSerial.h>

#define SDA\_PIN 21

#define SCL\_PIN 22

#define OLED\_RESET -1

Adafruit\_SSD1306 display(OLED\_RESET);

HardwareSerial gpsSerial(1); // Use Serial1 for ESP32

TinyGPSPlus gps;

void setup() {

  Serial.begin(115200);

  gpsSerial.begin(9600, SERIAL\_8N1, 16, 17);

  if (!display.begin(SSD1306\_I2C\_ADDRESS, SDA\_PIN, SCL\_PIN)) {

    Serial.println(F("SSD1306 allocation failed"));

    for (;;);

  }

  display.display();

  delay(2000);

  display.clearDisplay();

  display.setTextSize(1);

  display.setTextColor(SSD1306\_WHITE);

}

void loop() {

  while (gpsSerial.available() > 0) {

    if (gps.encode(gpsSerial.read())) {

      display.clearDisplay();

      display.setCursor(0, 0);

      display.print(F("Lat: "));

      display.println(gps.location.lat(), 6);

      display.setCursor(0, 10);

      display.print(F("Lon: "));

      display.println(gps.location.lng(), 6);

      display.setCursor(0, 20);

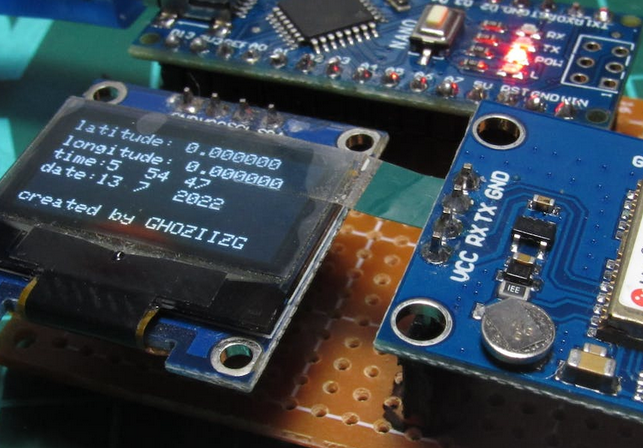
      display.print(F("Alt: "));

      display.println(gps.altitude.meters());

      display.display();

    }

  }



THANK YOU