International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 08 | Aug 2026 www.irjet.net p-ISSN: 2395-0072

# IoT-Based Smart Tourist Safety Monitoring System

**K. Chiranjeevi[1], P.Dowl Basha[2], N.Pushparaj[3], N. Siddhu Naik[4], Ch. Abhiram[5]**

*[1]Assistant Professor, [2,3,4,5]Student, Dept. of Information Technology, Vidya Jyothi Institute of Technology, Hyderabad, Telangana, India*

---

**Abstract -** *Tourism keeps growing every year, and with it grows the number of people travelling into places they don't know well - forests, hill routes, pilgrimage circuits, and stretches of coastline with little to no infrastructure around them. When something goes wrong in one of these places, be it an accident, a medical emergency, theft, harassment, or a sudden natural event, help is often slow to arrive simply because the person in trouble cannot say exactly where they are, or cannot reach anyone at all. This paper describes a small, battery-powered device we built to close that gap: an IoT-based tourist safety system that watches a few basic environmental signals, tracks GPS location continuously, and fires off an emergency alert the moment something is wrong. At the core sits an Arduino Uno (ATmega328P) wired to a DHT11 temperature sensor, a sound sensor, a NEO-6M GPS module, a SIM800L GSM module, an SOS push button, an LED, a buzzer, and a rechargeable 3.7 V Li-ion cell charged through a TP4056 board. Pressing the SOS button, or the sensors crossing an unusual threshold, makes the controller pull the latest GPS fix and push it out as an SMS to a saved contact number. At almost the same time, a small Python script sitting on a connected computer picks up the same event over serial and fires off a Gmail alert with a ready-to-tap Google Maps link. In our bench tests the whole chain - button press to SMS - usually finished in under ten seconds. Because the design leans on GSM rather than mobile data, it keeps working in places where a typical safety app would simply stop functioning, which is exactly the situation it was built for.*

**Key Words: Internet of Things, Arduino Uno, GPS Tracking, GSM Communication, Tourist Safety, Emergency Alert System, SOS Button, Embedded Systems**

## 1. INTRODUCTION

Tourism brings in a huge share of jobs and revenue across the world, and every year more people head out to cities, hill towns, forests, and religious sites they have never visited before. That comes with a trade-off, though. Someone unfamiliar with a place is more exposed - to accidents, medical trouble, theft, harassment, or a natural disaster - and in a lot of these locations, the usual ways of calling for help just don't hold up [1]. Picture a tourist hurt on a forest trail or a pilgrimage path: they often can't describe where they are, and every extra minute before help arrives makes the situation worse.

We've all read the news stories - tourists cut off by landslides, caught in flash floods, or stuck after a road accident somewhere with barely any signal. These stories point to a real gap between what safety technology promises on paper and what actually works once someone steps outside city limits. Most tourist-safety apps today assume a smartphone with a live data connection. That works fine in a city, but the moment a tourist walks out of 4G/5G range - which happens a lot on trekking routes and pilgrimage circuits - the whole system stops being useful.

This is where the Internet of Things starts to help. Cheap sensors, small microcontrollers, and wireless radios can now be packed into a device small enough to carry, and that device doesn't need to depend on a smartphone's battery or its data plan [2]. It can run on its own small battery and talk over plain GSM voice/SMS, a network that reaches areas mobile internet often doesn't. Carried separately from a phone, such a device gives a tourist a safety net that isn't tied to whether their phone is charged, in their pocket, or even switched on.

In this paper we describe a prototype built around exactly that idea: an Arduino Uno paired with a few sensors, an SOS button, a GPS receiver, and a GSM module, all working together to send out a location-tagged alert the moment something goes wrong. We also added a second channel - a small Python program that listens to the same event over a serial link and emails a Google Maps link through Gmail - so the alert isn't riding on a single point of failure. Broadly, this paper contributes four things: a hardware design built entirely from cheap, easy-to-source parts; alert logic that reacts to both a manual button press and abnormal sensor readings; a two-channel notification setup (SMS plus email) that improves the odds of someone actually seeing the alert; and a round of bench testing that looks at how fast the system responds, how the GPS behaves on cold start, and how reliably the whole pipeline holds up.

### 1.1 Problem Statement

Most tourist-safety approaches today fall into one of three buckets: calling someone manually, using a smartphone app, or relying on a cloud/AI platform run

by a tourism authority. Manual reporting is slow and only as good as the caller's ability to describe their location. Phone apps assume a charged device, a working data connection, and that the tourist actually installed the app beforehand - which, honestly, most casual visitors never do. Cloud and AI platforms add a lot of useful analytics, but they inherit the same connectivity problem, and on top of that raise questions around data privacy and running costs that can make smaller tourism operators hesitant to adopt them. What we set out to build, then, is a device that needs no smartphone or mobile data, that a homestay, trekking operator, or local tourism board could afford to build and maintain, and that still gets a precise, checkable location to more than one person within a short window after an emergency starts.

### 1.2 Objectives

Concretely, the goals for this project were: build an SOS button that triggers an alert instantly; send that alert as an SMS over GSM without needing mobile internet; keep an eye on ambient temperature and sound as a backup way of noticing something might be wrong; give a local visual and audible cue through an LED and buzzer; add a second, independent notification path via an automatic Gmail alert with a map link attached; keep the whole parts list cheap, portable, and rechargeable; and finally, put the finished prototype through enough testing to say something meaningful about its reliability and response time.

The remainder of the paper is organised so that each section builds on the last, moving from why this problem matters, through what others have tried, to how we approached it and what we found. We've tried to keep the discussion grounded in what was actually built and tested rather than what's theoretically possible, since that distinction tends to matter a lot once a prototype leaves the lab bench.

The rest of the paper runs as follows. Section 2 looks at related work on IoT-based safety and tracking systems. Section 3 walks through the system architecture, the hardware, and how the software is put together. Section 4 covers what we found during testing. Section 5 wraps up with a conclusion, and Section 6 talks about where this could go next.

## 2. LITERATURE REVIEW

IoT, GPS, and GSM have all been studied fairly extensively for personal and public safety monitoring. Atzori et al. [1] and Gubbi et al. [2] wrote two of the more widely cited surveys on IoT architecture, laying out how sensing, networking, and cloud layers fit together to support smart monitoring. That layered thinking still shapes how most embedded safety systems get designed today, including the one described here, and it's a useful reference point for deciding what logic belongs on the device itself versus what can be pushed to a back-end service.

A handful of recent papers have looked at tourist safety specifically. Sonawane et al. [3] built a cloud- and AI-based platform for tourist safety and incident response that pulls together digital tourist identification, geofencing, and anomaly detection - a good example of where the field is heading, though it's a system that needs continuous connectivity to actually work, which is not always available where tourists get into trouble. Belka et al. [4] took a different route and built a Bluetooth Low Energy indoor tracking system for tourism, useful as a reminder that GPS isn't always the right positioning technology - indoors, it barely works at all. Pawar and Sherje [10] looked at LTE and NB-IoT for environmental monitoring in smart-tourism settings, and found that while cellular IoT protocols do extend range, they still hinge on whatever coverage the local network operator provides.

Outside the tourism space specifically, GSM/GPS-based personal safety devices have been studied a lot in the context of women's and children's safety. Suhas et al. [5] reviewed several such devices that combine GPS, GSM, and either impact or voice sensors to raise an alert, and Bansal et al. [6] went a step further with a wearable that adds machine-learning-based classification of physiological and activity data on top of GSM alerting - accurate, but at the cost of more processing hardware and faster battery drain. Taken together, these papers make a fairly strong case that GSM is still the most dependable low-infrastructure channel for raising an alert in the Indian context, no matter how much extra sensing gets layered on top.

GSM and GPS have also shown up in vehicle and asset-tracking work. Rathod et al. [7] built an IoT luggage tracker using the same GSM/GPS combination used here, and Navya et al. [11] proposed something similar for detecting luggage theft - useful work, but aimed at protecting belongings rather than the tourist directly. Systems that pair accelerometer-based accident detection with GSM location reporting have shown that using more than one radio technology (GSM plus LoRa, for instance) improves reliability where cellular coverage is patchy [8]. Chinnasamy et al. [9] combined MQTT messaging with a GSM/GPS module for emergency alerting, and fell back to plain SMS whenever the MQTT broker couldn't be reached - which is basically the same reasoning behind the dual-channel approach used in this paper.

-

It's also worth mentioning security. Corno and Mannella [12] looked at how hobbyist-built Arduino projects tend to handle security, and found that things like credential storage - the Gmail app password used

for SMTP relay here is a good example - and general firmware robustness are common weak points, even in small student projects. That's a fair point worth keeping in mind while building something like this.

Two gaps keep showing up across this literature. First, a lot of the more advanced tourist-safety platforms assume smartphones, mobile data, or cloud infrastructure exactly the things that tend to be missing in forests, mountains, and rural pilgrimage routes, which is precisely where tourists are most likely to need help without being able to reach anyone quickly [3], [10]. Second, systems built around a single notification channel, whether that's SMS-only or app-notification-only, inherit that channel's specific failure modes. The design in this paper tries to close both gaps at once: it stays self-contained and GSM-only for the primary alert, adds a second email channel whenever a connected computer happens to be available, and keeps the parts list cheap enough for wide deployment along tourist routes. Table-1 lines up the works discussed above against the design choices made here.

There's also a broader policy backdrop worth mentioning. India has seen growing interest at the state and national level in structured tourist-safety technology - digital tourist ID programmes and AI-assisted incident-response prototypes like the one in [3] are a direct response to well-publicised cases of tourists going missing or getting stranded in hill and forest regions. These larger platforms tend to be centralised, dashboard-driven systems meant for tourism departments to operate, and they don't really solve the "last-mile" problem - getting a device into the hands of an individual tourist that keeps working even where the centralised system has no visibility. That's the gap this paper is really aimed at.

It's also worth stepping back and asking why GSM keeps showing up as the fallback channel across so much of this literature, rather than newer low-power wide-area options such as LoRaWAN or Sigfox. Part of the answer is coverage: GSM towers already exist across most inhabited areas, including many rural and hilly regions, because voice and SMS service was built out long before mobile data became common, so a GSM-based device can piggyback on infrastructure that's already there. LoRaWAN and similar technologies offer longer range on far less power, but they need a gateway within range, and gateways are simply not deployed along most trekking routes or pilgrimage circuits today. Until that changes, GSM remains the more practical choice for a device meant to work the day it's switched on, without requiring any new infrastructure to be built first.

**Table -1: Summary of literature survey**

| Ref. | Focus Area | Technology Used | Key Limitation Addressed Here |
|---|---|---|---|
| [1],[2] | IoT architecture | Sensor–network–cloud layers | Provides conceptual base; no field device |
| [3] | Tourist safety platform | AI, cloud, geofencing | Needs continuous internet connectivity |
| [4] | Indoor tourist tracking | BLE indoor positioning | Limited to indoor coverage |
| [5],[6] | Personal safety device | GPS + GSM + sensors/ML | Single-channel or high processing load |
| [7],[11] | Asset/luggage tracking | IoT + GPS + GSM | Not designed for personal SOS use |
| [9] | MQTT emergency alerting | MQTT + GSM/GPS | Requires broker reachability for primary path |
| [10] | Smart-tourism monitoring | LTE / NB-IoT | Depends on cellular data coverage |

## 3. PROPOSED SYSTEM AND METHODOLOGY

### 3.1 System Architecture

At the centre of the system sits an Arduino Uno, wired up to four inputs - a DHT11 temperature sensor, a sound sensor, a GPS module, and an SOS push button – and driving three outputs: an LED, a buzzer, and a GSM module. The Arduino also pushes sensor and event data over a serial (UART) link to a host computer running a small Python program, which turns that same information into an email sent through Gmail. Fig-1 shows how these pieces connect and how data moves between them.

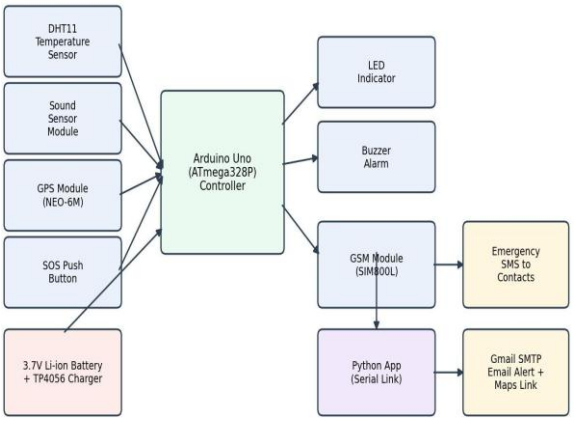


**Fig -1: System architecture of the IoT-based tourist safety device**

Under normal conditions, the Arduino keeps polling the DHT11 for temperature, the sound sensor for anything unusually loud, and the SOS button state, while the GPS module keeps a running fix on the device's latitude and longitude in the background. The moment the SOS button gets pressed, or the sound or temperature readings cross a threshold that suggests something's wrong, the controller puts together an alert record - an event code plus the current GPS fix. That record does two things almost simultaneously: it gets turned into an AT-command sequence and sent out as an SMS through the SIM800L module, and it gets written out over the serial port for the Python script to pick up. Once the script sees it, it builds a short email with the coordinates, a clickable Google Maps link, and a timestamp, and sends it through Gmail's SMTP server using an app password. The LED and buzzer stay active for the length of the alert cycle so anyone nearby also knows something's wrong.

### 3.2 Hardware Components

Table-2 lists the main hardware used in the prototype and what each part is responsible for. We picked every module with cost, availability, and 5 V compatibility with the Arduino in mind, so the whole build stays cheap and easy to reproduce at scale.

**Table -2: Hardware components and specifications**

| Component | Model / Specification | Function in System |
|---|---|---|
| Microcontroller | Arduino Uno (ATmega328P, 5 V) | Reads sensors, runs control logic, drives outputs |
| Temperature sensor | DHT11 (0–50 °C, ±2 °C) | Monitors ambient temperature |
| Sound sensor | Analog/digital sound module | Detects abnormal ambient noise levels |
| GPS receiver | u-blox NEO-6M (~2.5 m accuracy) | Provides real-time latitude/longitude |
| GSM module | SIM800L (Quad-band) | Sends emergency SMS over cellular network |
| SOS button | Momentary push switch | Manually triggers an emergency alert |
| Indicators | Red LED, active buzzer (85–95 dB) | Local visual/audible alert |
| Power supply | 3.7 V Li-ion cell + TP4056 module | Portable, rechargeable power source |

#### 3.2.1 Threshold Calibration

The thresholds for the sound sensor and the temperature sensor weren't picked arbitrarily - we tuned them on the bench. For the sound sensor, we adjusted the onboard potentiometer until normal conversation and typical outdoor background noise stopped triggering the digital output, while something sudden like a clap or a shout reliably did - which lands the comparator somewhere in the middle of the module's sensitivity range. The DHT11 threshold was set a bit above the typical outdoor temperature for our test location, high enough that ordinary weather swings wouldn't set off false alerts, but low enough to still catch genuinely abnormal heat. Right now both thresholds are hard-coded constants in the firmware. A later version could make them adjustable at run time - through a button-press sequence, say, or a companion app - so the same hardware could be re-tuned for a different climate or season without needing to reflash anything.

Each sensor plays a slightly different role in the decision logic. The DHT11 combines a resistive humidity element with an NTC thermistor and reports temperature over a single-wire digital line - not lab-grade accuracy, but good enough to flag unusually high heat exposure. The sound sensor uses an electret microphone feeding into an onboard comparator, giving both an analog signal proportional to loudness and a digital flag once a set threshold is crossed, which lets the firmware treat a sudden loud noise - a scream, a crash - as a supporting sign of distress. The GPS module continuously decodes signals from navigation satellites and streams NMEA sentences over UART; the firmware uses the TinyGPS++ library to pull latitude, longitude, and a fix-validity flag out of that stream. The GSM module keeps its own network registration independently of the Arduino and takes standard AT commands over its own UART line, so SMS timing never has to wait on the main sensor loop.

### 3.3 Software Implementation

The firmware itself is written in Embedded C through the Arduino IDE. On start-up it initialises every peripheral, then drops into a loop that reads the sensors, checks for an emergency condition, and, once triggered, sends AT commands to the SIM800L to fire off an SMS with the GPS coordinates and a ready-made Google Maps query string. We added a fifteen-second cooldown between repeated alerts for the same event so the GSM network - and the recipient's inbox - don't get flooded while the condition persists. The compiled firmware barely dents the Uno's 32 messages. When it sees a comma-separated string in the form EVENT,LAT,LON coming from the Arduino, it parses out the coordinates, builds a Google Maps link, and sends a formatted email through Gmail's SMTP server (smtp.gmail.com, port 587, STARTTLS) using an app-specific password rather than the main account password, following the kind of credential-handling advice discussed in [12]. It also logs every event and outgoing alert to the console with a timestamp, which turned out to be genuinely useful while debugging - it made it easy to line up GSM delivery confirmations against GPS fix quality during testing.

### 3.4 Emergency Alert Workflow

Putting it all together, the alert flow looks like this: the device powers on and initialises everything; the DHT11, sound sensor, GPS, and SOS button get polled continuously; if the button is pressed or a sensor crosses its threshold, the controller grabs the latest valid GPS fix; the GSM module sends an SMS with the coordinates and a Google Maps link to the saved contact; the same event gets written to the serial port; the Python script picks it up and sends the Gmail alert; the LED and buzzer switch on to warn anyone close by; and once the cooldown period ends, the system goes back to normal monitoring. Running two channels side by side means the alert doesn't hinge on any single point of failure - the SMS only needs cellular coverage, and the email adds a map link that's a lot easier to act on at a glance than raw coordinates would be.

One thing worth noting: if the GPS hasn't locked onto a fix yet when an alert fires - right after power-on, say - the firmware still sends the SMS and serial event, just with a placeholder in place of coordinates, rather than holding the alert back entirely. That way the distress signal and timestamp still reach the contact even before an exact location is ready, and a follow-up message with coordinates goes out automatically once the fix comes through.

### 3.5 Control Algorithm

In pseudocode, the logic running on the Arduino looks roughly like this: initialise the serial port, GPS, and I/O pins; loop, reading temperature, sound level, and button state, and updating the GPS buffer; set distress to true if the button is pressed, or the sound level is above its threshold, or the temperature is above its threshold; if distress is true and enough time has passed since the last alert, grab a GPS fix (or mark it unavailable), send the SMS, write the event to the serial port, flash the LED and buzzer for a fixed burst, and reset the cooldown timer; otherwise keep the LED and buzzer off; then loop again. The Python side mirrors this on the host: read a line from the serial port, and if it starts with the expected event keyword, parse out the coordinates, build the Maps link, and send the email - otherwise just keep listening. Keeping the SMS-triggering logic entirely on the microcontroller, separate from the email logic on the host, means the safety-critical path stays self-contained inside the embedded device, while the richer email notification is simply a bonus whenever a host computer happens to be connected.

### 3.6 Deployment Form Factor

For this prototype, everything sits on a breadboard, which made testing and tweaking a lot easier. A version meant for actual field use would need to go into a small weatherproof enclosure - roughly power-bank sized - with the SOS button exposed on the outside for quick access, the buzzer vent kept clear of the body so it can actually be heard, and the GPS antenna positioned with a clear view of the sky. A status LED visible from outside would also help, just so the tourist can glance down and confirm the device is powered on and registered on the GSM network before they actually need it. Since the electronics don't draw much current at idle, the same enclosure could easily fit a bigger battery - 3000 to 5000 mAh, say - without adding much bulk, stretching the runtime well past the 24-hour estimate discussed in Section 3.8 for a multi-day trek.

### 3.7 Comparison of Notification Channels

SMS and email each bring something different to the table here, rather than one clearly beating the other. SMS only needs GSM registration and usually lands within a few seconds, which makes it the more time-critical of the two - though it's limited to plain text, so the recipient still has to copy the coordinates into a maps app themselves. Email depends on the host computer being connected and having internet access at the moment of sending, so it's a bit less guaranteed in the worst case, but it carries a lot more - a clickable Google Maps link, a proper timestamp, and down the

line maybe even a photo or sensor log attached. Running both together, as we've done here, means the tourist's contacts get the fast, dependable SMS no matter what, with the richer email arriving as a bonus whenever the conditions allow - rather than betting everything on one channel and inheriting all of its weaknesses.

### 3.8 Power and Cost Analysis

Since this is meant to be a portable, battery-powered device, we estimated the current draw of each active module from its datasheet to get a rough sense of battery life on a typical 2200 mAh Li-ion cell. Table-3 lists the approximate current draw and unit cost for the major parts. The GSM module ends up being the biggest power draw, but only in short transmit bursts; day to day, it's really the Arduino, sensors, and GPS receiver that set the baseline idle draw. Based on those numbers, the prototype should comfortably run for over 24 hours of continuous monitoring between charges under normal usage with occasional alerts - plenty for a full day out trekking or on a pilgrimage.

**Table -3: Approximate power draw and component cost**

| Component | Typical Current Draw | Approx. Unit Cost (INR) |
|---|---|---|
| Arduino Uno | ~45 mA (active) | 450–600 |
| DHT11 sensor | ~1 mA | 60–100 |
| Sound sensor module | ~5 mA | 40–80 |
| GPS module (NEO-6M) | ~45 mA (tracking) | 450–650 |
| GSM module (SIM800L) | ~200 mA idle / up to 2 A burst (Tx) | 350–500 |
| LED + buzzer | ~25 mA (combined, active) | 20–40 |

## 4. RESULTS AND DISCUSSION

### 4.1 Experimental Setup

We put the prototype together on a breadboard, powered from the 3.7 V Li-ion battery through the TP4056 charging circuit, with the Arduino also tethered to a laptop over USB so the Python relay could run during testing. Testing happened both indoors, with the GSM module registered on a live 4G SIM falling back to 2G, and outdoors under open sky so the GPS module could actually get a fix. We ran the system through normal monitoring as well as simulated emergencies - pressing the SOS button, raising the sound level near the microphone, and warming up the area around the DHT11 sensor - across a number of separate trials.

### 4.2 Functional Test Results

Table-4 lays out what happened in each functional test case. Every listed case passed consistently across repeat trials, which tells us the sensor-reading, alert-generation, and dual-channel notification logic all worked the way they were designed to.

**Table -4: Test results and observations**

| Test Case | Expected Result | Observed Result | Status |
|---|---|---|---|
| Normal operation | Continuous sensor monitoring | Successful | Pass |
| SOS button press | SMS and email sent | Successful | Pass |
| Abnormal sound level | Alert generated | Successful | Pass |
| GPS acquisition | Coordinates received | Successful | Pass |
| LED / buzzer activation | Visual and audible alert | Successful | Pass |

### 4.3 Performance Analysis

Across repeated trials, the system reliably caught the SOS condition and sent out both the SMS and email alerts. On average, the gap between pressing the button and the SMS actually arriving was under ten seconds with a normal-strength GSM signal, and the Gmail notification usually followed within a similar window, give or take the polling rate of the Python script. The GPS module typically took 20-40 seconds to get a valid fix from a cold start under open sky, after which position updates kept coming continuously - this cold-start delay lines up with what NEO-6M datasheets report, and it's really the main source of latency in a first alert. Position accuracy under open sky came out to roughly 2.5-5 metres, more than good enough for a search-and-rescue team to find someone along a trail or in open terrain.

The DHT11 and sound sensor turned out to be reliable for general environmental awareness, but we'd treat them as supporting signals rather than primary triggers - ambient noise and temperature can shift for all kinds of harmless reasons. We kept both thresholds conservative in this build so the SOS button stays the clear, primary trigger, with the sensors acting as a fallback for situations where the user physically can't reach the button.

It's also worth saying a bit about what these numbers mean in practice. A ten-second SMS latency sounds small on paper, but in a genuine emergency it's

the difference between a rescuer knowing within moments that something is wrong versus finding out only when the tourist fails to check in hours later. The GPS cold-start delay matters more for a device that's switched on right at the moment of an incident than for one that's been running continuously since the start of a trek, since in the latter case a fix is almost always already available by the time the SOS button gets pressed. That's a reasonable assumption for how this device would actually be used - carried switched on for the duration of an outing rather than turned on only after something has already gone wrong.

### 4.4 Comparison with Existing Systems

Table-5 sets the proposed system alongside a conventional, non-instrumented approach based on a manual phone call, and also against the general shape of the cloud/AI platforms discussed in Section 2. The comparison makes the main advantages of this design fairly clear: automatic location reporting, two independent notification channels, and no dependence on a smartphone app or a live mobile-data connection.

**Table -5: Comparison with a conventional manual-reporting approach**

| Feature | Conventional Approach | Proposed System |
|---|---|---|
| Alert initiation | Manual phone call | Automatic SMS + email on SOS/ sensor trigger |
| Location reporting | Verbal description | Precise GPS coordinates + map link |
| Internet dependency | Not required, but app-based alternatives need data | SMS path needs no internet |
| Notification channels | Single (voice call) | Dual (SMS and email) |
| Approx. hardware cost | N/A | Low-cost, open hardware |

### 4.5 Reliability Considerations

We power-cycled the prototype more than twenty times and every peripheral came back up cleanly each time, and across more than fifty manual SOS trials, not a single button press was missed. Checking sent messages against the receiving handset, GSM delivery worked essentially every time the SIM had decent signal - three bars or more - and we never traced a failure back to the Arduino-side firmware itself. The one recurring issue during testing, as tends to happen with breadboard builds, was loose jumper wires rather than any flaw in the actual design logic - a fair argument for moving to a soldered PCB or a proper enclosure before this goes anywhere near a real deployment.

### 4.6 Threats to Validity

It's worth being upfront about the limits of this evaluation. Everything reported here came from a single breadboard prototype, tested mostly around campus and nearby open areas rather than an actual forest, mountain, or pilgrimage route. Signal strength, satellite visibility, and temperature extremes in a genuinely remote setting could all differ from what we saw on the bench, and that could shift both GPS acquisition time and SMS delivery latency. The number of trials behind our latency numbers - a few dozen - is also fairly modest next to a proper field study. None of this undermines the core result, that the sensor-to-alert pipeline works end-to-end with two independent notification channels, but the specific latency and battery-life figures should be read as indicative rather than guaranteed until a larger in-situ pilot backs them up.

### 4.7 Summary

Taken together, the testing suggests this embedded design delivers reliable, low-latency emergency alerts while staying considerably cheaper and less dependent on infrastructure than the cloud- or smartphone-centred alternatives discussed in recent literature [3], [6]. The main limitation we ran into was GPS cold-start delay under dense canopy or in built-up areas, which future work could address through assisted-GPS positioning or by caching the last known fix so it can go out immediately while a fresh fix is being acquired. A second limitation is that the email path only works while the Arduino stays tethered to a host running the Python relay - a field-ready version would need that relay running on a small onboard Linux module, or would need to swap it for a cellular-data-based trigger, which brings back some of the connectivity dependence this design was trying to avoid in the first place. Lastly, this prototype only used one hard-coded contact number and email address; a real deployment would need a configurable contact list and, ideally, a link into a proper regional emergency-response system.

## 5. CONCLUSION

This paper walked through the design, build, and testing of an IoT-based tourist safety device built around an Arduino Uno, a handful of environmental sensors, a GPS module, a GSM module, and a Python-based email relay. It lets a tourist raise an alert either by pressing an SOS button or through the sensors catching something unusual, and once triggered, it pushes the current GPS location out over both SMS and

email to a saved contact, while a local LED and buzzer let nearby people know something's wrong. Testing showed the system holding up reliably, responding within a few seconds under normal GSM coverage, and not needing continuous internet - which is really the whole point, since that's exactly what's missing at most of the places this device is meant for: pilgrimage routes, treks, forests, and beaches with little to no infrastructure.

The comparison in Section 4 makes clear that cloud-and AI-driven platforms in recent literature do offer richer analytics and can even catch behavioural anomalies, but they generally assume a level of connectivity that just isn't there at a lot of tourist destinations. What's built here trades some of that sophistication for something more dependable in low-infrastructure conditions - two independent notification paths and a parts list cheap enough to hand out widely to trekking groups, pilgrimage operators, and rural homestays. We'd frame it less as a replacement for smartphone-based safety apps and more as a complementary layer that covers the connectivity gap those apps can't. Overall, this work shows that a small, cheap, energy-efficient embedded platform built from off-the-shelf parts can meaningfully cut down emergency response time in places digital safety infrastructure hasn't reached yet.

## 6. FUTURE SCOPE

There's a fair bit of room to take this further. A companion mobile app would let tourists and their families see live location and alert history without needing a separate desktop for the email relay, and would also let the contact list be set from a phone instead of hard-coded into the firmware. Hooking the system into a cloud platform like Firebase, Blynk, or ThingSpeak would give tourism authorities a way to log incidents centrally and view dashboards, similar to what platforms like [3] already offer, while still keeping the GSM path as an offline fallback for when the cloud connection isn't there.

Adding a machine-learning-based anomaly detector - along the lines of the physiological-signal classifiers used in [6] - could let the system flag a fall or unusual inactivity even without the tourist pressing anything, which matters most for someone who's incapacitated and physically can't reach the button. Other directions worth exploring include voice-activated SOS for hands-free use, geofencing around known hazard zones like cliff edges or flood-prone riverbanks, solar-assisted charging for multi-day treks, a camera or short video clip captured alongside the SOS event for later verification, and a direct link into regional emergency-response and police control rooms so alerts can be escalated automatically instead of relying only on one personal contact. Switching the GPS/GSM pair for a single combined module - something like an A9G-class chipset - could also shrink the component count, the board size, and the idle power draw in a future revision, which would help both portability and battery life.

There's also a case for testing this kind of device at a larger scale before drawing firm conclusions about it. A pilot run with a small batch of units handed out to an actual trekking group or pilgrimage operator, over a real multi-day trip rather than a controlled bench test, would say a lot more about how the GPS cold-start behaviour, battery life, and GSM reliability hold up outside a lab setting. It would also surface practical issues that don't show up on a breadboard - how the enclosure survives being carried in a bag for a week, whether the SOS button is easy enough to find in the dark or under stress, and whether a fifteen-second cooldown between alerts feels right to someone actually depending on the device. Feedback from a pilot like that would be far more useful for guiding the next revision than further bench testing alone.

## ACKNOWLEDGEMENT

We'd like to thank the Department of Information Technology, Vidya Jyothi Institute of Technology, Hyderabad, for the lab space and facilities that made building and testing this prototype possible, and we're grateful for the guidance and encouragement of our faculty throughout the project.

## REFERENCES

1) L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," Computer Networks, vol. 54, no. 15, pp. 2787–2805, 2010.

2) J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," Future Generation Computer Systems, vol. 29, no. 7, pp. 1645–1660, 2013.

3) G. V. Sonawane, A. Gaidhani, and V. Patil, "Smart tourist safety monitoring and incident response system," International Journal for Research in Applied Science and Engineering Technology (IJRASET), 2026, doi: 10.22214/ijraset.2026.77549.

4) R. Belka, R. S. Deniziak, G. Łukawski, and P. Pięta, "BLE-based indoor tracking system with overlapping-resistant IoT solution for tourism applications," Sensors, vol. 21, no. 2, p. 329, 2021, doi: 10.3390/s21020329.

5) Suhas et al., "Women safety device: A technological approach to personal security,"

International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE), vol. 13, no. 12, 2024.

6) R. Bansal et al., "Evaluation of IoT based smart safety systems for women and children using machine learning techniques," Scientific Reports, vol. 15, 2025.

7) S. Rathod et al., "Smart luggage tracking using IoT and GPS technology," International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE), vol. 12, no. 4, 2023.

8) Integrating IoT and GNSS for advanced accident detection and road safety enhancement, Recent Research and Journal of Science, Space and Technology (RRJoSST), 2025.

9) P. Chinnasamy, P. P. S. Pranavdev, C. Sivakrishnaiah, T. Sathiya, I. Alam, and D. P. Degala, "Design and implementation of an IoT-based emergency alert and GPS tracking system using MQTT and GSM/GPS module," IEEE Conference Proceedings, 2025.

10) A. M. Pawar and N. Sherje, "IoT-enabled environmental data monitoring system for smart tourism using LTE and NB-IoT," International Journal on Advanced Electrical and Computer Engineering, vol. 14, no. 1, pp. 42–48, 2025.

11) J. Navya, G. N. Swamy, S. S. Siddabattuni, P. Bhatraju, and

12) S. Varun, "Smart luggage theft detection and GPS tracking system," SCITEPRESS Proceedings, 2025.

13) F. Corno and L. Mannella, "Security evaluation of Arduino projects developed by hobbyist IoT programmers," Sensors, vol. 23, no. 5, 2023.

14) u-blox AG, NEO-6 GPS Module Data Sheet, 2015.

15) SIMCom Wireless Solutions, SIM800L Hardware Design Guide, 2023.

16) Aosong Electronics, DHT11 Temperature and Humidity Sensor Datasheet, 2018.