GNSS SPOOFING DETECTION USING PYTHON AND MACHINE LEARNING TECHNIQUES

A Real-Time Lightweight Solution Using the Haversine Formula

Presentation By :-

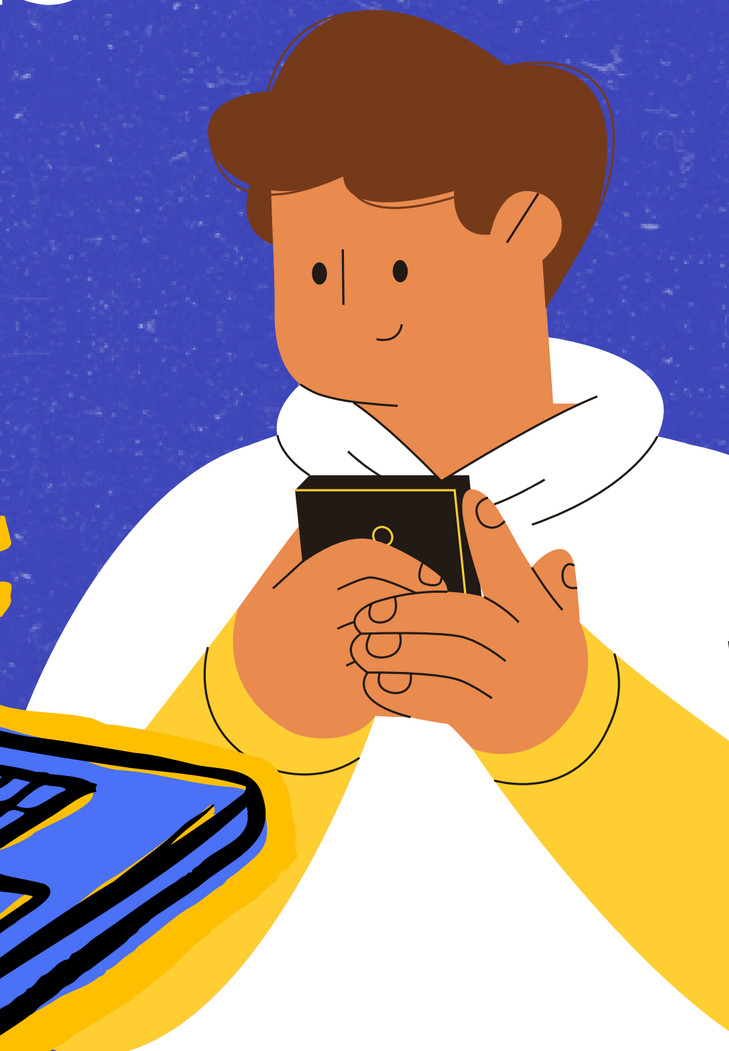
Nandini Choudhari

Sanika Desai

Ishwari Barve

INSTITUTION

MULUND COLLEGE OF COMMERCE



Introduction

- GNSS (GLOBAL NAVIGATION SATELLITE SYSTEM) IS WIDELY USED FOR POSITIONING, NAVIGATION, AND TIMING IN SYSTEMS LIKE DRONES, AUTONOMOUS VEHICLES, AND DEFENSE.
- GNSS SPOOFING INVOLVES SENDING FAKE GPS SIGNALS TO MISLEAD A RECEIVER.
- SPOOFING ATTACKS CAN RESULT IN INCORRECT LOCATION DATA, POTENTIALLY CAUSING ACCIDENTS, LOSS OF ASSETS, OR OPERATIONAL FAILURE.
- OUR SOLUTION AIMS TO DETECT THESE SPOOFING ATTEMPTS IN REAL-TIME USING PYTHON AND A SIMPLE BUT EFFECTIVE SPEED-BASED METHOD.

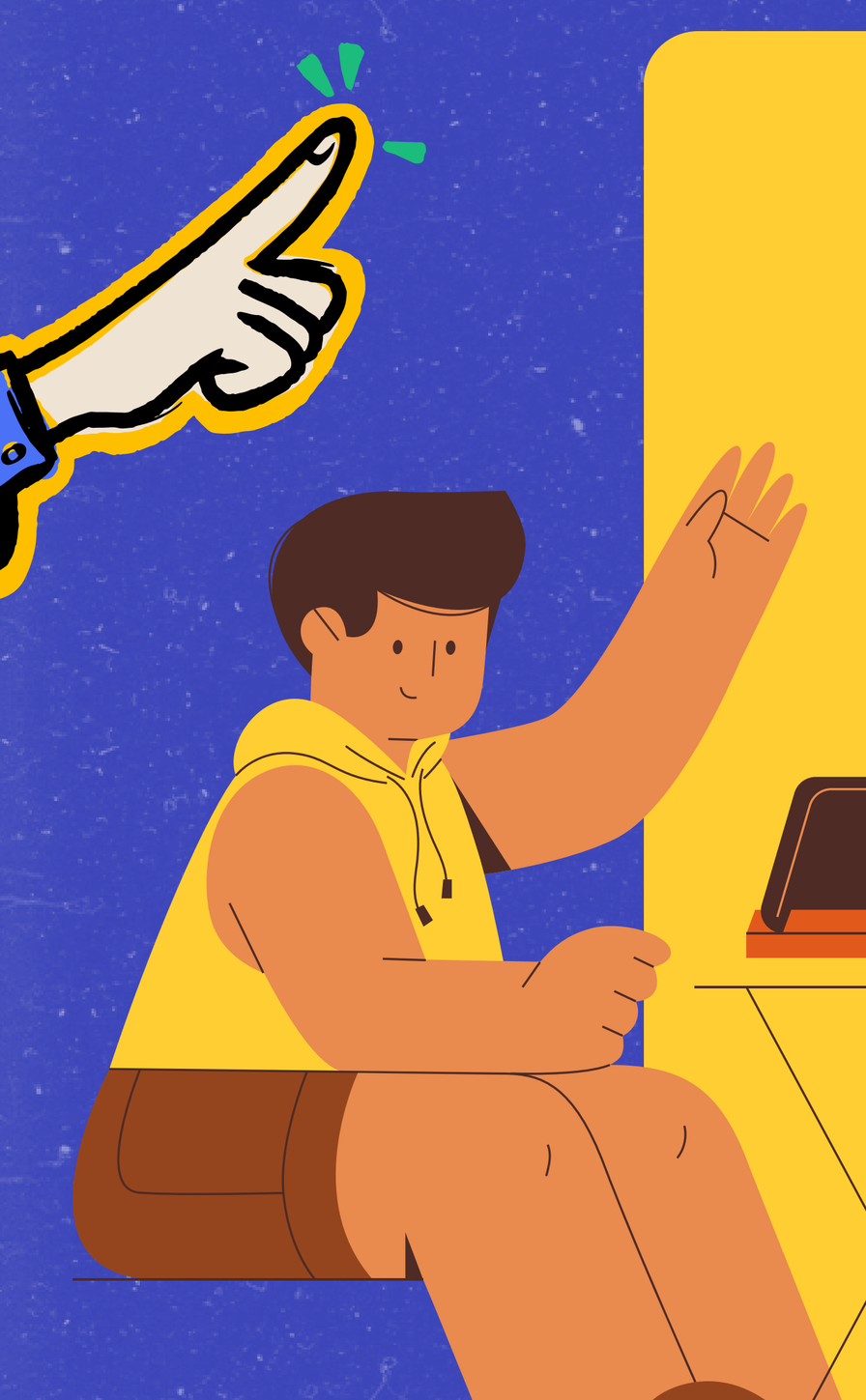
PROBLEM STATEMENT & OBJECTIVE

Problem Statement:

- GNSS systems are vulnerable to spoofing attacks, which can manipulate positioning data.
- This is a serious threat to autonomous vehicles, transportation, and defense.
- Existing detection methods are either hardware-dependent, costly, or complex.

- 🎯 Objective:
- To develop a Python-based tool that detects GNSS spoofing in real-time.
- It uses the Haversine formula to calculate speed between GPS points.
- If speed exceeds a threshold (e.g., 300 km/h), it flags it as a potential spoofing attempt.
- The goal is to offer a lightweight, cost-effective, and portable solution.

LITERATURE REVIEW



📚 Key Insights from Existing Research:

- Early methods used signal strength and multi-frequenc checks, but were hardware-intensive.
- Recent research focuses on machine learning (SVMs, neur networks) for spoofing detection.
- ML methods show high accuracy, but require large dataset and are computationally heavy.
- Most existing tools are not optimized for real-time detectio on low-power devices.

🔍 Our Approach Stands Out:

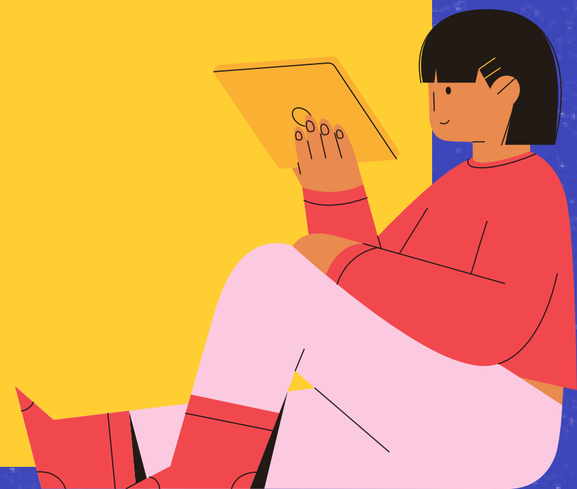
- A simpler, faster, and lightweight method using just spee calculation.
- Can run on mobile or embedded systems without extr hardware.

RESEARCH METHODOLOGY

- 🧪 Step-by-Step Process:
- Collect GNSS Data in NMEA format (includes time, latitude, longitude).
- Parse the data using Python.
- Apply the Haversine formula to compute distance between two consecutive GPS points.
- Calculate speed using distance and time difference.
- Check if speed > 300 km/h → Flag as a possible spoofing attempt.
- Log spoofing attempts for further analysis.
- ✅ Benefits of This Approach:
- Lightweight and portable
- Real-time processing
- No extra hardware needed

TOOL IMPLEMENTATION

- 🛠️ Technologies Used:
- Python Libraries:
- math for Haversine distance calculation
- datetime for calculating time difference
- open() for reading NMEA GPS files
- 🔄 Implementation Logic:
- Parse GPS data from NMEA file.
- Calculate distance between consecutive GPS points using Haversine formula.
- Compute speed from distance and time difference.
- Flag speeds > 300 km/h as potential spoofing.
- Log spoofing attempts for further review.
- Key Features:
- Lightweight and real-time.
- No external hardware required.
- Portable and cost-effective solution.



CODE SNIPPET

```
IMPORT MATH
FROM DATETIME IMPORT DATETIME

DEF HAVERSINE(LAT1, LON1, LAT2, LON2):
    R = 6371  # EARTH'S RADIUS IN KM
    DLAT = MATH.RADIANS(LAT2 - LAT1)
    DLON = MATH.RADIANS(LON2 - LON1)
    A = MATH.SIN(DLAT/2)**2 + MATH.COS(MATH.RADIANS(LAT1)) * /
        MATH.COS(MATH.RADIANS(LAT2)) * MATH.SIN(DLON/2)**2
    RETURN R * 2 * MATH.ATAN2(MATH.SQRT(A), MATH.SQRT(1 - A))

DEF DETECT_SPOOF(FILE_PATH):
    WITH OPEN(FILE_PATH) AS F:
        LINES = F.READLINES()
```

```
LAST_LAT, LAST_LON, LAST_TIME = NONE, NONE, NONE
FOR LINE IN LINES:
    IF "$GPRMC" IN LINE:
        PARTS = LINE.SPLIT(',')
        LAT = FLOAT(PARTS[3][:2]) + FLOAT(PARTS[3][2:]) / 60
        LON = FLOAT(PARTS[5][:3]) + FLOAT(PARTS[5][3:]) / 60
        TIME_STR = PARTS[1]
        TIME_OBJ = DATETIME.STRPTIME(TIME_STR[:6], "%H%M%S")

        IF LAST_LAT:
            DIST = HAVERSINE(LAT, LON, LAST_LAT, LAST_LON)
            TIME_DIFF = (TIME_OBJ - LAST_TIME).SECONDS OR 1
            SPEED = (DIST / TIME_DIFF) * 3600
            IF SPEED > 300:
                PRINT(F"🚨 SPOOFING ALERT AT {TIME_OBJ}: SPEED = {SPEED:.2F} KM/H")

        LAST_LAT, LAST_LON, LAST_TIME = LAT, LON, TIME_OBJ
```

LOGIC FLOW OF THE TOOL

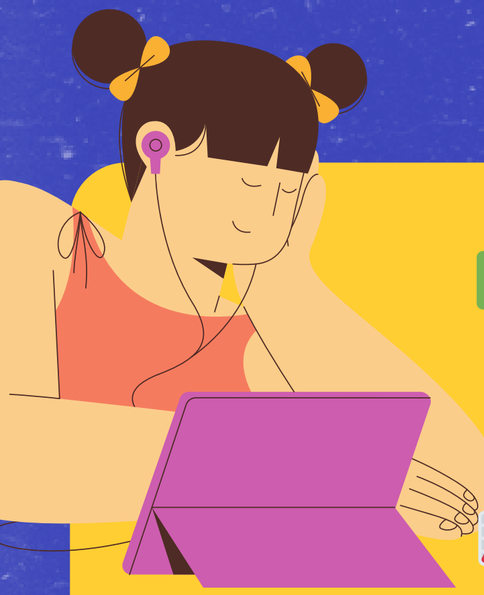
🧭 GNSS SPOOFING DETECTION – STEP-BY-STEP FLOW

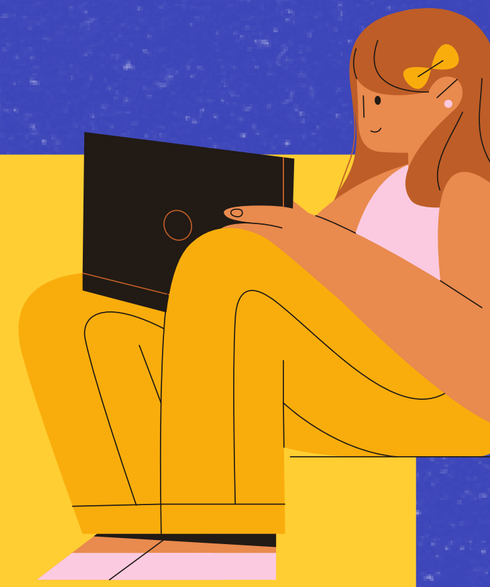
1. READ GPS DATA
2. → LOAD NMEA FORMAT DATA FROM A .NMEA FILE.
3. PARSE GPS COORDINATES
4. → EXTRACT LATITUDE, LONGITUDE, AND TIME FROM $GPRMC LINES.
5. CALCULATE DISTANCE
6. → USE THE HAVERSINE FORMULA TO COMPUTE DISTANCE BETWEEN TWO CONSECUTIVE P
7. COMPUTE SPEED
8. → DIVIDE DISTANCE BY TIME DIFFERENCE TO GET SPEED IN KM/H.
9. SPOOFING CHECK
10. → IF SPEED > 300 KM/H → FLAG AS POTENTIAL SPOOFING ATTEMPT.
11. LOG OR ALERT
12. → PRINT OR STORE THE ALERT WITH TIMESTAMP AND CALCULATED SPEED.

✅ KEEP THE SLIDE MINIMAL AND USE ICONS OR ARROWS FROM CANVA FOR VISUAL FLOW.

WOULD YOU LIKE A VISUAL-STYLE FLOWCHART INSTEAD OF BULLET POINTS FOR THIS SLIDE?

RESULTS & OBSERVATIONS





✅ Test Data:

- 1,000 GPS data points (mix of real and spoofed signals)
- Threshold speed: 300 km/h

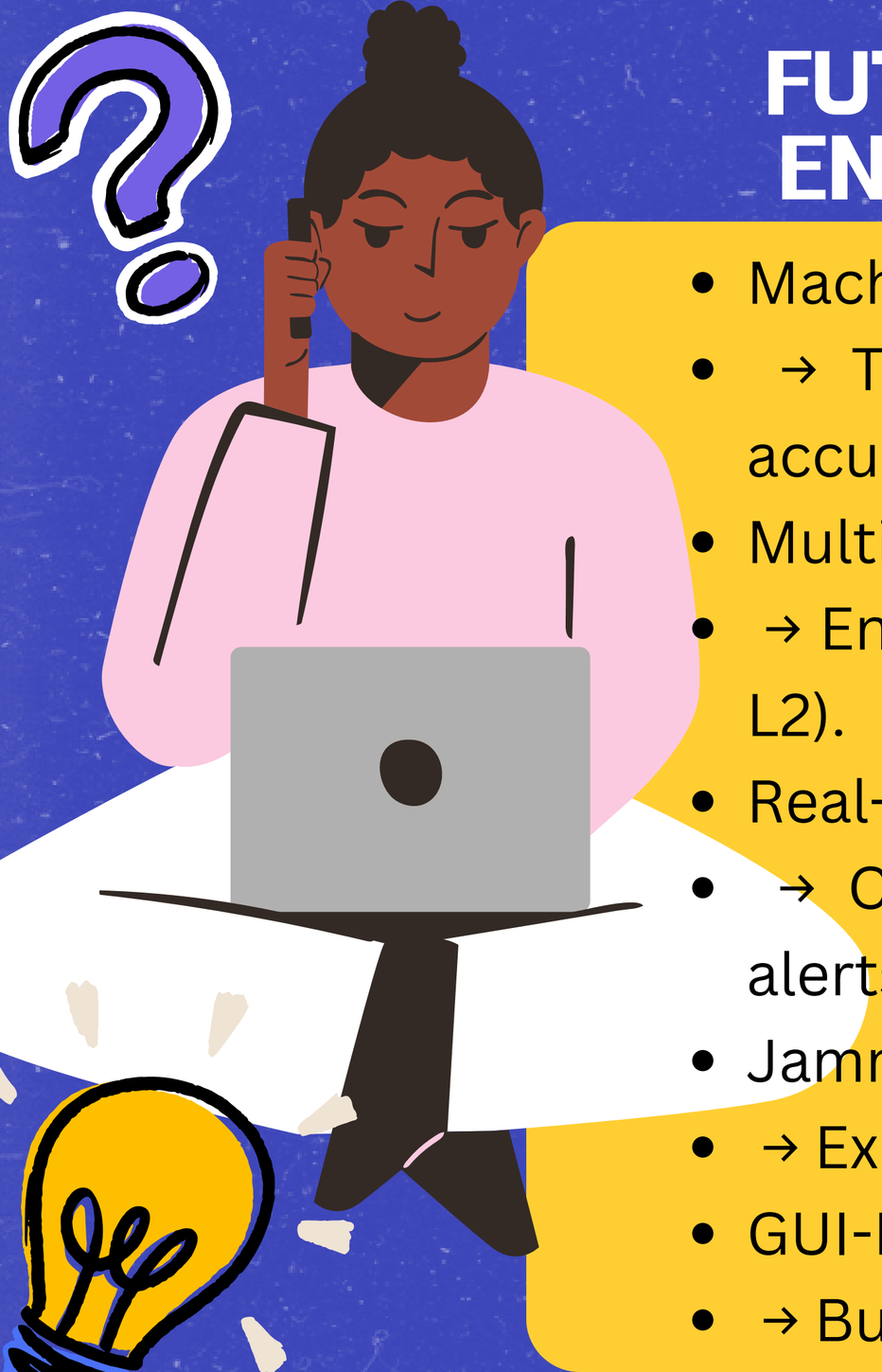
📈 Tool Performance:

- Detection Accuracy: ~95%
- False Positives: Minimal during normal movement
- Real-time Alerts: Successfully flagged suspicious data

🔍 Example Case:

- Sudden jump in GPS location
- Speed = 700+ km/h → Tool flagged as 🚨 Spoofing

FUTURE SCOPE & ENHANCEMENTS



- Machine Learning Integration
- → Train models to learn spoofing patterns for better accuracy.
- Multi-Frequency GNSS Support
- → Enhance detection by using more GNSS bands (e.g., L1, L2).
- Real-Time Hardware Integration
- → Connect with GPS devices to provide live spoofing alerts.
- Jamming & Interference Detection
- → Expand to identify jamming or signal interference too.
- GUI-Based Application
- → Build a user-friendly app for non-technical users.


THANK
YOU