令和 7 年度　特別研究報告書

スマートフォンを用いた
緊急車両位置推定システムの提案

龍谷大学　先端理工学部　知能情報メディア課程

Y220219　近藤　悠輝

指導教員　三好　力　教授

内容梗概

近年、緊急車両の接近に気づかず、進路妨害が発生するケースが社会問題となっている。本研究では、一般車両や歩行者に対し、緊急車両の接近を早期かつ確実に通知することを目的とし、スマートフォンの位置情報と音量データを活用した位置推定システムを提案する。

提案システムは、複数の端末から得られるサイレン音量を重みとした加重平均法により緊急車両の位置を推定し、さらにマップマッチング処理を適用することで、GPS 測位誤差の影響を補正する。

シミュレータを用いた定量的評価の結果、端末数が 7 台以上の環境下において推定精度が安定し、特に 20 台以上の中密度環境においては、提案手法が比較手法（単純平均法）を上回る推定精度を実現した。また、時速 30km 以下の低速走行時には数メートル以内の誤差で追従可能であり、交差点進入時や渋滞通過時など、支援の必要性が高い場面において極めて高い実用性を発揮することが確認された。さらに、GPS 環境が悪化する都市部を想定した耐ノイズ性試験においても、音量情報によるフィルタリング効果により精度の劣化を最小限に抑える高いロバスト性が実証された。

本研究の成果は、既存のスマートフォンインフラを活用しつつ、低コストかつ広範囲に展開可能な緊急車両支援システムの実現に寄与するものである。

目次

1. はじめに

1.1 研究背景

人命救助や火災鎮圧など、一刻を争う事態に対応するため、救急車や消防車といった緊急車両は、道路交通法に基づき緊急走行を行うことが認められている。[1]しかし、その緊急走行中における一般車両との交通事故は依然として発生しており、重要な社会課題の一つとなっている。救急振興財団による全国調査の報告によれば、調査に回答した消防本部だけでも合計 1,659 件の緊急走行中の救急車事故が発生しており、救急出動 10 万件あたりの事故発生リスクは 13.9 件に上るとされている[2]。これらの事故は、緊急搬送の遅延を引き起こすだけでなく、一般市民や救急隊員の安全を脅かす深刻な事態につながりかねない。

こうした事故が発生する要因は複合的であるが、一般車両のドライバーが緊急車両の接近に気づきにくい環境が形成されつつある点が指摘できる。第一に、近年の自動車は技術革新により静粛性や遮音性が大幅に向上しており、車外で発生しているサイレン音が車内まで届きにくくなっている。第二に、カーオーディオで音楽を聴きながら運転するなど、車内環境によって外部の音が認知しにくくなるケースも多く、サイレン音の認知を一層困難にしている。

さらに、ドライバーの運転経験も事故要因として挙げられる。運転経験の浅い初心者ドライバーは、経験豊富なドライバー（熟達者）と比較して、運転中の視覚探索において狭い範囲に注視を集中させる傾向があるとの研究報告がある[3]。このため、ミラー等で後方や側方を確認する頻度が低下し、そこから接近する緊急車両の発見が遅れがちになる。また、交通事故総合分析センターの統計によれば、若年ドライバーの事故の人的要因として「安全不確認」や「発見の遅れ」が多くを占めており[4]、緊急車両の接近という非定常的な状況において、瞬時に適切な回避行動（減速、左側への寄車など）を判断・実行することが困難であると考えられる。

現在、もし緊急車両の正確な位置情報（GPS 情報）がリアルタイムで一般車両に共有されれば、カーナビゲーションシステムやスマートフォンアプリを通じて、ドライバーへ直接的な注意喚起や回避指示を行うことが可能となり、事故防止に大きく寄与すると期待できる。しかし現状では、緊急車両の位置情報は、通信の秘密やプライバシー保護の観点[4]、および運用上の様々な理由から、一般のドライバー向けには開示されていない。

1.2 研究の目的

本研究は、緊急車両の GPS 情報が一般開示されていないという現状の制約の中で、緊急走行中の事故を未然に防ぐことを目的とする。

その目的を達成するため、本研究では、一般市民が所持する多数のスマートフォンから収集した「位置情報」および「サイレンの音量情報」に着目する。これらの情報をリアルタイムで集約・解析することにより、緊急車両の現在位置を高い精度で推定し、さらにはその進行方向や速度から未来の到達位置を予測するシステムを提案する。
具体的には、サイレン音量が大きい地点のスマートフォンの位置情報をクラスタリングすることで緊急車両の存在を検出し、音量レベルの分布と時間的変化を追跡することで、その移動経路を割り出すアルゴリズムを開発する。さらに。本システムにより推定された緊急車両の位置・未来位置情報を、周辺を走行する一般車両のドライバーに対して視覚的・聴覚的に提供することで、早期の認知と的確な回避行動を支援する。これにより、緊急車両の円滑な通行を確保するとともに、緊急走行に関わる交通事故の抜本的な低減を目指す。

2. 既存技術

2.1 トヨタ ITS Connect

トヨタ自動車が推進する ITS (Intelligent Transport Systems: 高度道路交通システム) を活用した安全運転支援システムである。

2.1.1 概要

ITS Connect は、従来の車載センサー（カメラやレーダー）では捉えきれない情報を、通信によって取得しドライバーに通知するシステムである[6]。本研究の目的に関連するものとして、緊急車両（対応送信機を搭載した救急車など）の存在を検知し、ドライバーに早期の認知と回避行動を促す機能が含まれている。

2.1.2 V2V・V2I 通信

ITS Connect は、車両とインフラ (路車間通信: V2I) または車両と車両 (車車間通信: V2V) が直接通信を行う。この通信には、ITS 専用周波数として割り当てられている 760MHz 帯の電波が使用される。この周波数帯は、情報通信の遅延が少なく、高速移動体同士でも安定した通信が可能であり、携帯電話網とは独立した専用の通信チャネルである。この通信により、対応する車載器（受信機）を搭載した車両は、対応する送信機を搭載した緊急車両のおおよその位置・進行方向・距離の情報を受信できる。

2.1.3 導入効果

ITS Connect は、特に交差点周辺での事故低減に効果を示している。日本の交通事故のうち約 4 割は交差点および交差点付近で発生しているとされる[7]。ITS Connect に搭載される機能のうち、「緊急車両存在通知」に関しては、ITS Connect 搭載車のユーザーの**90% 以上が「(緊急車両の) 退避に役立つ」**と評価している[8]。この機能の普及により、緊急車両の円滑な通行が促進され、搬送時間の短縮が期待されている。

2.2 NICT「救急車の位置情報をサイレン音に載せて周囲のカーナビに表示」

国立研究開発法人情報通信研究機構 (NICT) が研究開発した、サイレン音を利用して緊急車両の位置情報を伝達する技術である[9]。

2.2.1 概要

緊急車両が発するサイレン音に、デジタル情報（電子透かし）を重畳（スーパーインポーズ）させる技術である。周囲の車両に搭載されたカーナビやスマートフォンが、マイクを通じてこのサイレン音を拾い、重畳されたデジタル情報をデコード（解読）することで、緊急車両の正確な位置や進行方向を把握し、ドライバーに通知する仕組みを目指している。

2.2.2 音響電子透かし

中核となる技術は「音響電子透かし」である。これは、音声データ（この場合はサイレン音）の特性を人間の耳には知覚できないレベルでわずかに変化させ、そこに特定のデジタル情報（緊急車両の GPS 座標や進行方向データなど）を埋め込む技術である。既存の音響媒体であるサイレン音そのものを情報伝達チャネルとして利用する点が特徴であり、緊急車両側には GPS 情報と連動してサイレン音に情報を埋め込む専用の送信装置（エンコーダー）が必要となる。

3. 提案手法

3.1 既存技術における問題点

「２．既存技術」で述べた技術は、緊急車両の接近をドライバーに通知するという目的において有効性を持ちつつも、社会全体への普及には大きな課題を抱えている。
ITS Connect（V2X 通信）における課題は、その導入コストと普及の前提条件にある。このシステムは、情報を送信する緊急車両側と、情報を受信する一般車両側の双方が、760MHz 帯の専用通信機（車載器）を搭載していることが動作の前提となるため、どちらかの普及が進まなければ、システム全体の有効性が高まらない。 また、車載器の搭載は車両購入時のコスト増加要因となり、現状では搭載車種が限定的である。さらに、緊急車両を保有する消防や警察といった公的機関側にも、全車両への送信機導入コストが発生する。このように、社会インフラとして機能するまでに莫大なコストと長い時間を要する点が最大の課題である。
NICT の音響電子透かし技術における課題も、同様に送信側のコストと、伝達媒体の信頼性にある。 第一に、ITS Connect と同様、緊急車両側に情報をエンコード（重畳）するための専用装置を設置する必要があり、導入コストと普及の課題を抱えている。 第二に、より本質的な問題として、「音」を情報伝達媒体として利用する物理的な脆弱性が挙げられる。高速走行時の風切り音やロードノイズ、トンネル内の反響、豪雨の音などは、マイクが音響透かし情報を正確にデコードする妨げとなる。また、「1.1 研究背景」でも述べた近年の自動車の高い遮音性やカーオーディオの使用環境も、正確な情報受信を困難にする要因となる。
以上のように、既存の主要な技術は、専用の車載器や送信機といった「専用ハードウェア」の普及を前提としており、それが導入コストと普及速度のボトルネックとなっている。

3.2 提案手法

3.1 で述べた既存技術の課題、すなわち「専用ハードウェア（車載器や送信機）の導入コストと普及の遅れ」という根本的な問題を解決するため、本研究では一般に広く普及しているスマートフォンをセンシングデバイスとして利用するシステムを提案する。
本システムは、特定の送信機を搭載していない緊急車両であっても、その存在を検知することを目的とする。これを実現するため、多数のスマートフォン（クライアント）から「位置情報（GPS)」と「サイレン音量」のデータをリアルタイムで収集し、それらの情報を中央サーバーで集約・解析する。

これは、V2X や音響透かしのような専用機器が情報を「発信」するアプローチとは異なり、多数の市民が持つスマートフォンをセンサーノードとする「クラウドソーシング型」の位置推定アプローチである。
この手法により、専用インフラの整備や車載器の購入を待つことなく、アプリケーションの普及のみで、緊急車両の接近をドライバーに警告するシステムの構築を目指す。

3.3 システム構成と仕様

提案するシステムは、大きく分けて「クライアントアプリケーション」と「中央サーバー」の 2 つのコンポーネントで構成される。
システムの一連の流れは以下の通りである。

(1) クライアント（スマートフォン）は、マイクでサイレン音を検知し、その音量（RMS）と自機の位置情報（GPS）を中央サーバーへ送信する。
(2) 中央サーバーは、多数のクライアントから送られた情報を集約し、「予測モデル」を用いて緊急車両の現在位置の推定と未来位置の予測を行う。
(3) クライアントは、中央サーバーが公開する「未来位置」を定期的に監視（ポーリング）する。
(4) クライアントは、自機の位置と緊急車両の未来位置との距離を計算し、その距離が一定の閾値を下回った場合にドライバーへ警告（アラート）を発する。

3.4 クライアントアプリケーション

クライアントアプリケーションは、利用者のスマートフォン上で動作し、主に「サイレンの検知」と「音の大きさの測定」と「ドライバーへの警告」という 2 つのセンシング機能を実行する。その計測結果に基づき、必要に応じてドライバーへの警告を行う。

(1) サイレンの検知
スマートフォンのマイクから入力された音声信号に対し**FFT（高速フーリエ変換）**を実行し、サイレン音に特有の周波数帯を検出する。FFT は、式 (3.1) に示されるように、時間領域の信号x_nを周波数領域のスペクトルX_kに変換する計算手法である。

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i\frac{2\pi}{N}kn} \tag{3.1}$$

日本のサイレン音の特性に基づき、検知対象の周波数帯を 700Hz～1500Hz の範囲に設定するのが良いと考えられる。

(2) 音の大きさの測定

サイレン音の音量の推移は、緊急車両とスマートフォンとの物理的な距離を推定するための重要な指標となる。本研究では、音の振幅（大きさ）を定量化するため、音声信号の **RMS（Root Mean Square: 実効値）**を測定する（式 3.2）。

$$x_{rms} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} \tag{3.2}$$

ここで、nはサンプル数、x_iは各サンプルの振幅を示す。このx_{rms}の値（音の大きさ）は、サイレン検知時の位置情報と共に中央サーバーへ送信される。

(3) ドライバーへの警告

ドライバーへの警告は、上記で取得したデータをサーバーに送信し、帰ってきた予測位置と、スマートフォンの GPS を用いた現在位置を用いる。

警告の判定処理は、スマートフォン端末のクライアント側で実行される。クライアントアプリは、サーバーから受け取った予測位置と、GPS から取得したリアルタイムの現在位置との間の距離を算出し、この距離が設定された警告閾値を下回るか否かを検証する。警告条件が成立した場合、ドライバーに対して即時かつ効果的な注意喚起を行うため、振動、聴覚アラート、および視覚的な画面配色変化を組み合わせたマルチモーダルな警告を発動する。これにより、ドライバーは危険が迫る前に必要な対応時間を確保できる。

以下に図 3.1 アプリケーションの UI を示す。

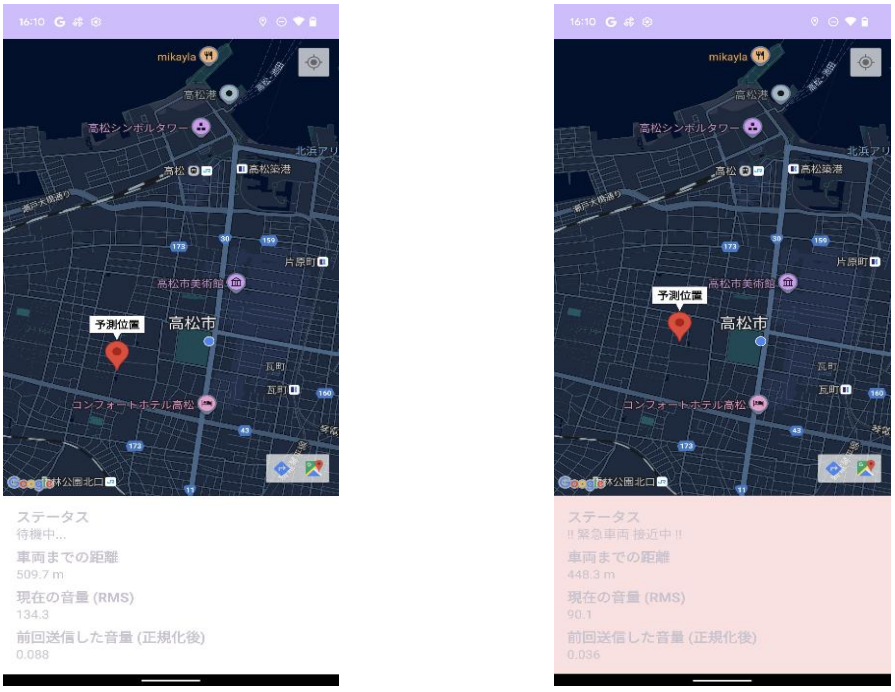

図 3.1 アプリケーション UI

利用者がいる位置は、青のポイント、緊急車両の予測位置は、赤のポイントでプロットされる。また、緊急車両が一定より遠くにある場合は、左の画像のようになっており、警告閾値を下回ると右の画像のように、画面を赤色に変化させ警告を行う。

3.5 予測モデル

中央サーバーは、収集したデータを基に「予測モデル」を実行し、緊急車両の現在位置と未来位置を算出する。このモデルは以下の 3 ステップで構成される。

3.5.1 中心位置の推定

サーバーは、直近の一定時間内に受信した全データ（n 個）を収集する。単純な位置の平均では、緊急車両から遠い（音量が小さい）データのノイズに影響されてしまう。
そこで、音量が大きい（＝緊急車両に近い）スマートフォンの情報（lat_i,lon_i）をより重視するため、音量（W_i）を重みとした加重平均を用い、現在の中心位置（$\overline{lat}, \overline{lon},$）を推定する（式 3.3, 3.4）。

$$\overline{lat} = \frac{\sum_{i=1}^{n}(lat_i \cdot W_i)}{\sum_{i=1}^{n} w_i} \tag{3.3}$$

$$\overline{lon} = \frac{\sum_{i=1}^{n}(lon_i \cdot W_i)}{\sum_{i=1}^{n} w_i} \tag{3.4}$$

実際には、実環境における音響検知においては、車種ごとの車体遮音性能の違いや、窓の開閉状態、さらにはスマートフォン個々のマイク感度や指向性といったハードウェア性能の差異により、観測される絶対音量は大きく変動する。したがって、本来であれば単一の音量閾値や絶対値を用いるのではなく、各ノードにおける時系列的な音量変化を解析し、相対的な接近度を算出する手法が望ましい。しかしながら、本研究の主目的は、位置情報と音量情報を組み合わせた加重平均アルゴリズムの基礎的な有効性を検証することにある。そのため、本シミュレーションにおいては問題を単純化し、全ての車両および端末が同一の音響特性を有し、理想的な減衰モデルに従ってデータが観測されるものと仮定した。

3.5.2 ベクトル計算による外挿

次に、推定した中心位置の時系列データから、未来の位置を外挿（予測）する。 まず、直近の移動傾向を把握するため、「現在位置（$P_{current}$）」（例：最新 2 秒間の平均中心位置）と「過去位置（P_{past}）」（例：7 秒前から 5 秒前までの 2 秒間の平均中心位置）を定義する。 この 2 点から、過去 5 秒間の移動ベクトルVを算出する（式 3.5）。

$$V = P_{current} - \mathrm{P_{past}} \tag{3.5}$$

このベクトルV（直近の進行方向と速度）を現在の位置$P_{current}$に足し合わせることで、4秒先の未来位置（P_{future}）を予測する（式 3.6）。

$$P_{future} = P_{current} + V \tag{3.6}$$

3.5.3 マップマッチングによる位置修正

前節で述べた位置予測モデル（加重平均および等速直線運動モデル）によって算出された予測位置は、GPS 誤差や予測ベクトルの変動により、必ずしも道路上に位置するとは限らない。特に、建物が密集する都市部やカーブの多い経路においては、予測位置が道路外へ逸脱する（オーバーシュート）問題が発生する。 そこで本システムでは、道路ネットワーク情報を用いたマップマッチング処理を導入し、幾何学的な整合性を担保する。
具体的な処理手順は以下の通りである。
（1）道路ネットワークの取得
OpenStreetMap (OSM) 等の地図データベースから、対象エリアの道路網データをグラフ構造（ノードとエッジ）として取得する。
（２）最近傍エッジの探索
予測位置 P(x_p, y_p)に対し、道路ネットワークを構成する全てのエッジ（道路区間）の中から、幾何学的な距離が最も近いエッジ$E_{nearest}$を特定する。
エッジEは、始点$A(x_1, y_1)$と終点$B(x_2, y_2)$を結ぶ線分として定義される。点Pと線分ABの距離$d(P, AB)$は、以下のように計算される。
ベクトル AB と AP を定義し、点 P から直線 AB への射影係数 t を内積を用いて算出する。

$$t = \frac{AP \cdot AB}{||AB||^2}$$

この t の値により、最短距離 d は以下のように場合分けされる。
$0 \le t \le 1$の場合距離は、点 P から直線 AB への垂線の長さとなる。外積（クロス積）の大きさを用いて算出される。
$t < 0$の場合距離は、点 P と始点 A とのユークリッド距離となる。
$t > 1$の場合距離は、点 P と終点 B とのユークリッド距離となる。
システムは、全ての道路エッジに対してこの距離 d を計算し（実際には R-tree 等の空間インデックスを用いて探索範囲を絞り込む）、d が最小となるエッジを選択する。

（3）直交射影によるスナップ

特定された最近傍エッジ$E_{nearest}$（始点 A、終点 B）に対して、予測位置 Pを移動（スナップ）させた補正位置$P'(x', y')$を算出する。

前項で求めた射影係数 tを用いることで、補正位置P'は以下のようにベクトル方程式で求められる。

$$OP' = OA + t \cdot AB$$

ただし、線分の端点を超える場合（$t < 0$または$t > 1$）は、それぞれ端点 A または Bを補正位置とする。

本システムでは、この$P'(x', y')$を最終的な「マップマッチング後の予測位置」として採用し、再び緯度・経度に戻して出力する。

この処理により、横方向の誤差を最小化し、ユーザーに対して「道路上を走行している」という自然な位置情報を提示することが可能となる。

4.評価実験

本章では、提案システムの有効性を検証するために行った評価実験について述べる。 まず、提案手法の根幹となる「音量による距離推定」の妥当性を実機を用いた予備実験により確認する。その上で、多様な交通状況や環境要因を考慮した詳細な性能評価を行うために構築したシミュレーション環境および実験条件について述べる。

4.1 予備実験（スマートフォンのマイクによる距離推定の検証）

4.1.1 概要

提案手法（加重平均法）では、「音源（緊急車両）に近い端末ほど大きな音量を観測する」という物理特性を前提として重み付けを行っている。この仮定が一般的なスマートフォンにおいて成立することを確認するため、実機を用いた予備実験を行った。
具体的な方法としては、スマートフォンのマイクを音源（サイレン音）から 1m,5m,10m,20m の各距離に配置し、取得される音量データ（デシベル値または正規化された強度）の変化を測定した。

4.1.2 結果と考察

測定の結果、距離の増加に伴って観測される音量値が減衰する傾向が確認された。環境ノイズや反射音による揺らぎはあるものの、距離と音量の間に負の相関関係が認められたことから、音量情報を位置推定の「重み」として利用する本手法のアプローチは妥当であると考えられる。

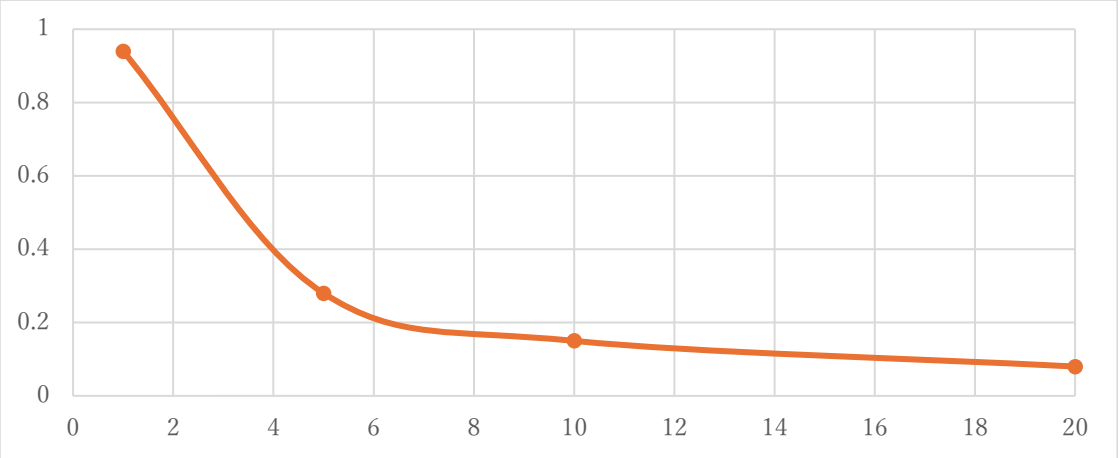

図 4.1　音量データ

4.2 実験概要

この章では、予測モデルの評価実験を行う。本研究が提案する「スマートフォンの位置情報と音量データを用いた緊急車両位置推定システム」の実用性を検証するためには、多様な交通状況や通信環境における性能評価が不可欠である。
特に、以下の要素はシステム性能に大きな影響を与えると考えられる。

- 端末密度がその場に居合わせたユーザー（観測点）の数が、推定精度にどのような影響を与えるか。
- 移動速度が緊急車両の走行速度の変化に対し、システムがどの程度の追従性とリアルタイム性を維持できるか。
- 観測ノイズが 都市部のビル街など、GPS 測位精度が著しく低下する環境下でも、システムが機能するか。

これらを検証するために数十台規模の移動端末を用いた実地実験を行うことは、同一条件下での再現性の確保や、安全性の観点から極めて困難である。例えば、GPS の誤差環境を人為的に制御したり、緊急車両を一定速度で何度も走行させたりすることは現実的ではない。
そこで本研究では、これらの環境要因を自由に制御可能なシミュレータを構築し、計算機実験による定量的評価を行うこととした。これにより、提案手法（加重平均法）と単純平均を用いた予測モデル(音声データなし)の比較、およびマップマッチング処理の有効性を、統計的に信頼できるデータ数を用いて検証する。

4.2.1 実験環境

本研究におけるシミュレータ、位置推定サーバー、および評価プログラムは、プログラミング言語 Python を用いて実装した。 システムの構成要素と使用した主要なライブラリを表 4.1 に示す。
位置情報の管理および通信には、非同期処理に優れた Web フレームワークである FastAPI と、軽量なリレーショナルデータベースである SQLite を採用し、リアルタイムに近いデータ処理環境を構築した。
地図情報の処理には OSMnx および NetworkX を使用した。これにより、OpenStreetMap (OSM) から高松市内の道路ネットワークを取得し、グラフ構造として扱うことで、緊急車両の走行ルート生成（最短経路探索）およびマップマッチング処理を実現している。
また、評価指標（XTE, ATE）の算出における幾何学計算には Shapely を、大量の時系列データの集計・統計処理には Pandas および NumPy を用いた。

表 4.1　使用ライブラリ

名称	本研究における主な用途
Python 3.x	システム全体の実装
FastAPI / Uvicorn	スマートフォンからの位置情報受信、予測 API の提供
SQLite3	検知データおよび予測結果の蓄積
OSMnx	道路ネットワークの取得、グラフ構造化
NetworkX	ノード間の最短経路探索（正解ルート生成）
Shapely	座標変換、点と線の距離計算（マップマッチング、XTE/ATE 算出）
Pandas / NumPy	時系列データの操作、誤差（MAE/RMSE）の統計処理
Folium	実験結果の地図上へのプロット（可視化）
Requests	シミュレータからサーバーへの HTTP 通信

4.3 シミュレータの仕様

本研究では、Python を用いて構築した位置情報シミュレータにより、仮想的なスマートフォン端末からのデータ送信を再現した。シミュレータは、正解ルート上を移動する緊急車両に対し、以下のロジックに基づいて観測データを生成する。

4.3.1 データ生成プロセス

各タイムステップ（1.0 秒間隔）において、車両周辺に設定された台数分の仮想端末を配置し、以下のデータをサーバーへ送信する。

(1) 位置情報：端末の「真の位置」に対し、GPS 測位誤差を重畳した座標。

(2) 音データ：緊急車両と端末の「真の位置」間の距離に基づき、距離の二乗に反比例する減衰モデルを用いて算出した数値（0.0〜1.0）。 なお、本シミュレーションでは、音量データには誤差を含めず、位置情報（GPS）のみに誤差が含まれるモデルを採用した。これにより、「音量は正しい（近距離を示唆している）が、位置情報はズレている」という、加重平均法にとって判断が困難な状況を意図的に再現し、アルゴリズムのロバスト性を検証する。

4.3.2 誤差モデル

端末の配置および GPS 誤差の生成には、一様分布ではなく正規分布（ガウス分布）を採用した。

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

現実世界において、道路上の歩行者や車両の分布、および電子機器の測定誤差は平均値周辺に集中する傾向があるため、極端な外れ値が均等な確率で発生する一様分布よりも、正規分布の方が現実の特性を適切に近似できると考えられるためである。

4.3.3 パラメータ設定

正規分布のパラメータである標準偏差（σ）は、統計的な信頼区間（2σ範囲内にデータの約 95.4%が含まれる特性）に基づき、以下の通り設定した。

(1) 端末の物理的散らばり（σ_{phys}）

緊急車両（正解位置）を中心として、端末が物理的にどの程度の範囲に存在するかを表すパラメータである。

- 設定値：σ_{phys}= 7.5m（2σ =15.0m）
- 根拠：緊急車両を中心として左右±15m（合計幅 30m）の範囲に 95%の端末が存在することを意味する。一般的な都市部の道路幅（片側 2〜3 車線＋歩道）や、沿道の建物内にいるユーザーを想定した場合、この範囲設定は妥当であると考えられる。

(2) GPS 測位誤差（σ_{gps}）

各端末が観測する位置情報の誤差を表すパラメータである。本実験では環境に応じた 2 つの条件を設定した。

- 標準条件: σ_{gps}=7.5m（2σ =15.0m）

一般的なスマートフォン GPS の精度は屋外で数メートル程度であるが、都市部におけるマルチパス（ビル反射）等の影響を考慮し、95%が半径 15m 以内に収まる精度を「標準的な都市環境」と定義した。

- 悪条件: σ_{gps}=15.0m（2σ =30.0m）

高層ビル街や悪天候など、測位条件が著しく悪い環境を想定した設定である。

(3) 比較手法間での条件統一

「加重平均法（WA）」と「単純平均法（SA）」の比較において、公平性を担保するため、最終的に生成される報告位置の統計的な散らばりが一致するように調整を行った。

- WA の場合：「物理的散らばり」と「GPS 誤差」を個別の正規分布として生成し、合算する。

$$Position_{wa} = TruePos(\sigma_{phys}) + Error(\sigma_{gps})$$

- SA の場合：音量情報を利用しないため、物理位置と GPS 誤差を区別する必要がない。したがって、誤差の加法性に基づき、合成標準偏差σ_{total}を用いた 1 回の正規分布生成で位置を決定した。

$$\sigma_{total} = \sqrt{{\sigma_{phys}}^2 + {\sigma_{gps}}^2}$$

標準条件（σ_{phys}= 7.5m, σ_{gps}=7.5m ）の場合、σ_{total} ≈10.6m となる。

これにより、両手法に入力される位置データの空間的な分布特性を統計的に完全に一致させた上で、音量情報の有無による推定精度の差異を検証可能とした。

4.4 評価手法

本節では、シミュレーションにより得られた予測データの精度を定量的に評価するための手法および指標について述べる。

4.4.1 評価におけるデータ処理

本提案システムの位置予測モデルは、移動平均ベクトルを用いて「4.0 秒先の未来位置」を推定するよう設計されている。したがって、予測精度を正当に評価するためには、予測データ生成時の時刻 t における正解位置と比較するのではなく、予測対象である未来時刻 t+4.0 における正解位置と比較を行う必要がある。
そこで本評価では、予測データのタイムスタンプ t に対し、正解データのタイムスタンプ t + 4.0 を対応させることで、未来予測の成否を検証可能な状態とした。また、移動平均フィルタの初期化に伴う立ち上がり期間（シミュレーション開始直後の不安定な挙動）を評価から除外するため、開始から 7 秒間（t=0〜6）のデータを破棄し、システムが定常状態に達した t=7 以降のデータを評価対象とした。

4.4.2 評価指標

位置推定の正確性およびシステムの実用性を多角的に検証するため、本研究では以下の 5 つの指標を用いて評価を行う。

(1) 平均絶対誤差(MAE)

予測位置と正解位置のユークリッド距離（直線距離）の平均値である。システム全体の位置精度の代表値として用いる。

(2) 二乗平均平方根誤差(RMSE)

誤差の二乗平均の平方根である。MAE に比べて大きな誤差（外れ値）の影響を強く受けるため、予測の安定性を測る指標として用いる。

(3) 横方向誤差(XTE)

正解ルート（道路中心線）から予測位置までの最短垂直距離である。 マップマッチング処理が正しく機能し、車両が道路上に位置づけられているかを評価するための幾何学的な指標である。

(4) 縦方向誤差(ATE)

正解ルート上における、正解位置と予測位置の進行方向の距離差である。 予測が「遅れている（ラグがある)」か、あるいは「進みすぎている（過剰予測)」かを示す時間的な指標

である。本システムにおいては、移動平均処理に伴う位相遅れの影響を確認するために用いる。

(5) ヒット率

予測結果が実用上の許容範囲内に収まっているデータの割合を示す指標である。本研究では以下の 2 つの基準を設けた。

- XTE ヒット率**:** 横方向誤差が道路幅相当（5m）に収まった割合。
- ATE ヒット率**:** 縦方向の遅れが、予測時間（4.0 秒）の移動距離相当=(速度/3.6)*4.0（例: 40km/h 時は約 44m）以内に収まった割合。これは、システムが「現在時刻よりも未来の位置」を表示できているかどうかの判定基準となる。

4. 5 検証項目と評価方法

構築したシミュレータにおいて、以下のパラメータを段階的に変化させることで、各特性を個別に評価する。

1. 密度特性の検証

目的：群衆の規模（データ量）と推定精度の関係性を明らかにする。

方法：車両周辺（半径 30m 以内）に存在するアクティブな端末数を 5 台（過疎）から 40 台（過密） まで変化させ、平均誤差（MAE）および横方向誤差（XTE）の推移を計測する。

2. 追従性能の検証

目的：車両速度がシステムの予測ラグ（遅延）に与える影響を確認する。また、オーバーシュートによる横方向誤差（XTE）をマップマッチングでどこまで修正できるかを明らかにする。

方法：車両の移動速度を 10km/h（徐行）から 60km/h（緊急走行） まで変化させ、縦方向誤差（ATE）と横方向誤差（XTE）の推移を計測する。

3. 耐ノイズ性の検証

目的：GPS 精度が悪化する悪条件下でシステムが正常に動作するか確認する。

方法：各端末に付与する GPS 誤差の標準偏差（σ）を、7.5m（標準的な都市部） と 15.0m（ビル街等の悪条件） の 2 パターンで設定し、精度の劣化度合いを比較する。

4. マップマッチングの効果検証

目的：幾何学的な補正処理が最終的な位置精度に与える寄与度を算出する。

方法：上記すべての実験において、マップマッチング処理の 「適用あり」と「適用なし」の両方のデータを取得し、横方向誤差（XTE）の改善幅とヒット率（道路内収束率）の変化を比較する。

5. 実用性の検証

目的：本システムの実用的な位置推定精度を達成するために最低限必要となる利用者数（端末密度）を特定する。

方法：緊急車両周辺（半径 30m 以内）のアクティブな端末数を 1 台から 15 台 まで段階的に変化させ、平均誤差（MAE)、横方向誤差（XTE)、および縦方向誤差（ATE）の推移を計測する。また、制度の変曲点を探し本システムの実用化における最低要件を算出する。

以上の検証を行うために用いた具体的な実験パラメータ設定を表 4.2 に示す。

表 4.2　シミュレータのパラメータ

検証項目	端末数(台)	速度（km/h）	GPS 誤差(m) 単純平均/加重平均
密度特性	5, 10, 15, 20, 30, 40	40	10.6/7.5
追従性能	15	10, 20, 30, 40, 50, 60	10.6/7.5
耐ノイズ性	10	40	10.6, 16.8/7.5, 15.0
実用性	1〜15(1 刻み)	40	7.5（加重平均のみ）
マップマッチング	上記のパラメータにおいて行う		

5.実験結果と考察

5.1 密度特性の結果

図 5.1「 MAE_m」より、すべての手法において、端末数の増加に伴い MAE が減少する傾向が見られた。サンプル数が増えることでランダムな GPS 誤差が相殺され、真値（車両位置）への収束が進んだためと考えられる。
図 5.2「SA、WA における MAE と XTE」より、手法間を比較すると、低密度環境においては、SA が WA よりも高い精度を示した。 これは、データ数が少ない状況下では、WA が個別の端末（特に GPS 誤差の大きい高音量端末）の影響を過度に受けやすいためであると考えられる。 一方、端末数が 20 台を超えると WA の精度が SA を上回り、音量による重み付けが有効に機能し始めた。そして、端末数が 40 台に達すると、SA と WA の差はわずかとなり、ほぼ同等の精度を示した。十分なデータ数が確保できる環境下では、単純な SA でも十分にノイズが除去されるため、WA による複雑な重み付けの優位性は相対的に小さくなることが示唆された。
このことから、提案手法（WA）は一定以上の端末密度が確保された環境において、その真価を発揮すると言える。
図 5.3「XYE のヒット率（5m 以下)」,5.4「ATE のヒット率（44.4m 以下)」より ATE のヒット率は、端末密度に関わらず全条件で約 90%以上の高い値を維持した。これは、本システムが、低密度環境下においても時間的な予測能力を堅牢に維持できていることを示している。一方で、XTE は密度低下に伴い悪化したことから、データ量の減少は、縦の誤差よりも、横の誤差対してより敏感に影響を与えるという特性が明らかになった。
図 5.4 より、マップマッチング（MM）適用時の ATE ヒット率が、未適用時に比べてわずかに低下する傾向が見られた。原因として、誤マッチングによる位置後退や射影による幾何学的遅延の影響が考えられる。すなわち、MM は XTE を劇的に改善する一方で、縦方向に関しては、誤マッチング等の副作用によりわずかながら遅延を助長する可能性があることが示唆された。 しかしながら、低下幅は数ポイント程度であり、システム全体の実用性を損なうものではない。

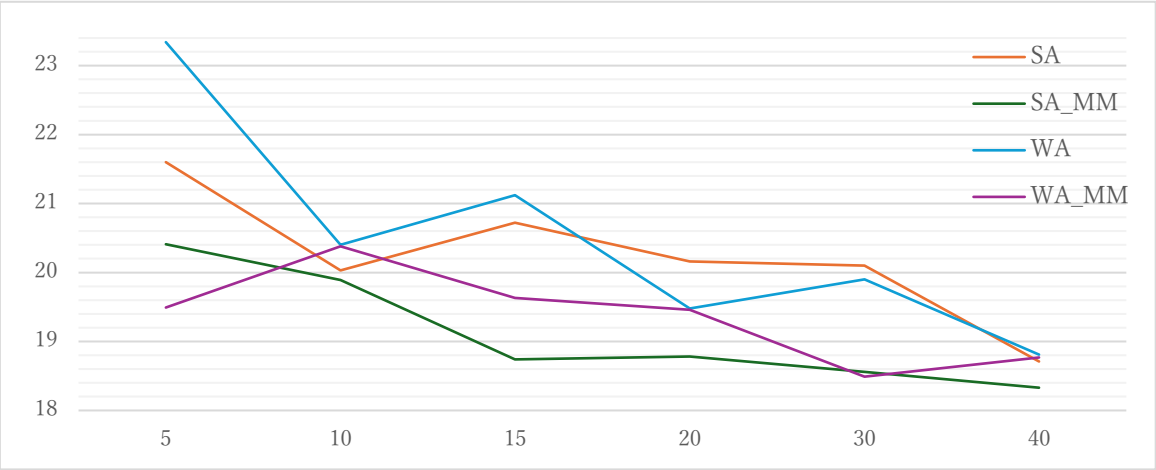

図 5.1　MAE_m

図 5.2　SA、WA における MAE と XTE

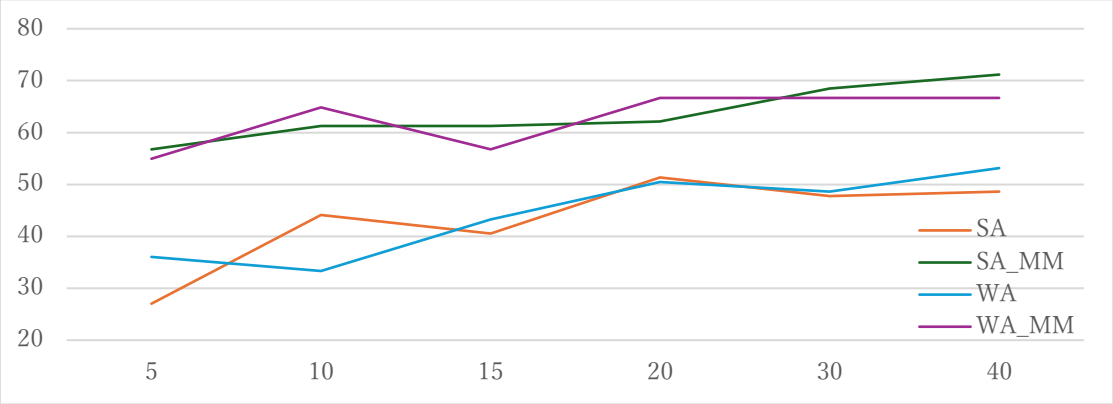

図 5.3　XYE のヒット率（5m 以下）

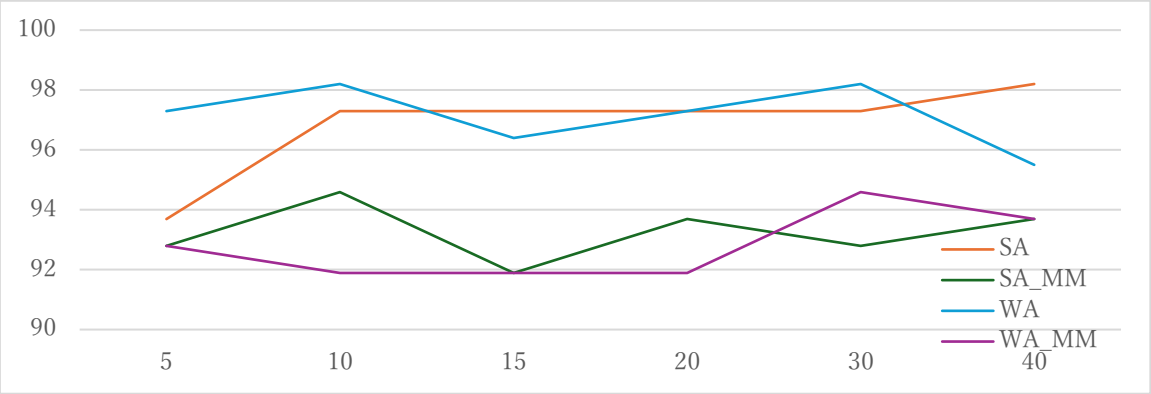

図 5.4　ATE のヒット率（44.4m 以下）

5.2 追従性能の結果

図 5.5「ATE」より、速度の上昇に比例して ATE が線形に増加する傾向が確認された。主な要因は、カーブ走行時における「オーバーシュート」にあると考えられる。図 5.6「WA__MM における 10km/h と 60km/h の曲がり角」でも確認できる。 本システムが採用する等速直線運動モデルは、カーブにおいても直進しようとするため、速度が速いほどカーブの外側へ大きく逸脱する。この逸脱した予測位置を正解ルート上にマップマッチングする際、幾何学的に進行方向手前側へ射影される「距離の損失」が発生する。 したがって、観測された ATE の増大は、システムの反応遅れではなく、高速走行時のオーバーシュートに起因する幾何学的な誤差であると考えられる。

図 5.7「XTE」,5.8「XTE のヒット率」より、XTE においては SA と WA の差は限定的であったが、ヒット率においては WA が SA を上回る傾向が見られた。

これは、SA が偶発的なノイズの影響で大きく逸脱するケースが含まれるのに対し、WA は音量情報によるフィルタリング効果により、極端な逸脱を抑制し、安定して道路近傍を推測できていることを示唆している。

図 5.7 より、XTE ヒット率の変化に着目すると低速域（10〜20km/h）においては、マップマッチングを用いることでヒット率が 85%〜90%と極めて高く、渋滞や徐行運転のような状況下ではほぼ完璧に道路上を追従できていることが示された。

一方、速度が上がるにつれてヒット率は低下し、60km/h 時には 43.66%となった。これは前述のオーバーシュートの影響により、カーブ区間で道路幅を逸脱するケースが増加したためであると考えられる

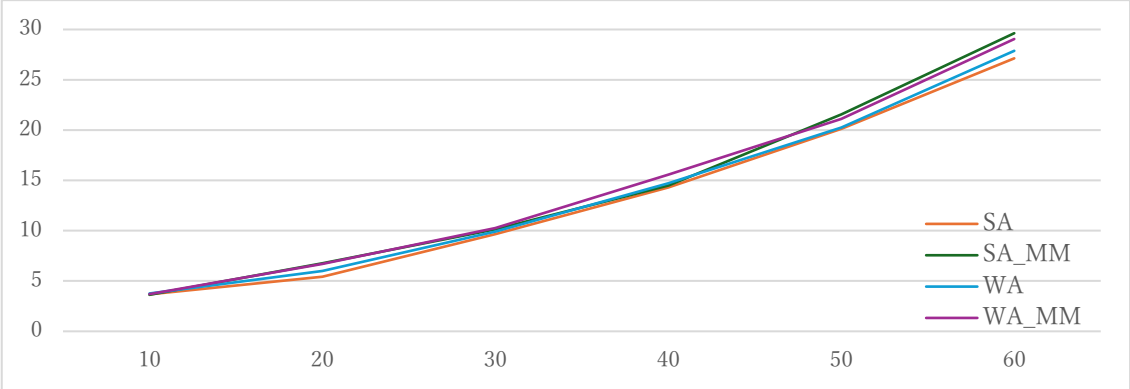

図 5.5　ATE

図 5.6　WA__MM における 10km/h と 60km/h の曲がり角

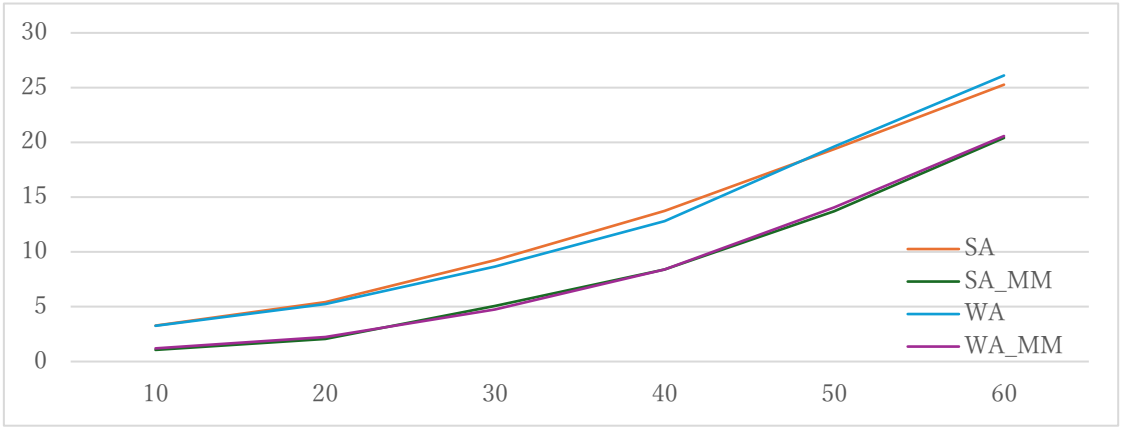

図 5.7　XTE

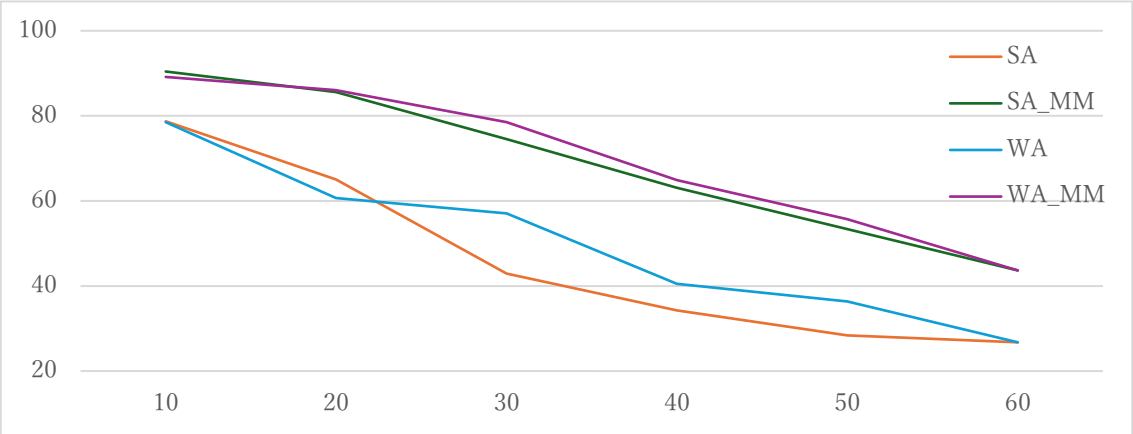

図 5.8　XTE のヒット率

5.3 耐ノイズ性

表 5.1「GPS の誤差における比較」より、SA と WA の比較において、顕著な差が確認された。SA は、GPS 誤差の増大に伴いヒット率が 20.7 ポイントも大幅に低下し、精度の崩壊が見られた。これは、SA が全ての端末の誤差を等価に扱ってしまうため、外れ値の影響を直接受けて予測位置が道路から大きく逸脱したためである。

一方、WA のヒット率低下は わずか 4.5 ポイントに留まった。絶対値こそ低いものの、環境が悪化しても性能を維持し続けることが示された。これは、WA が音量情報を利用して、信頼性の低い遠方のデータ（ノイズ）の重みを低減し、予測位置の拡散を抑制した結果であると考えられる。

XTE ヒット率では WA が優位性を示した一方、MAE および XTE における SA が同等以上の精度を記録した。平均値（MAE/XTE 平均）という指標は、1 件の大きな予測ミスがあるだけで、全体の数値を大きく下げてしまうという特徴がある。

また、WA は、高音量端末を重視することで通常時のヒット率を向上させるが、高音量かつ高誤差の特異な端末が存在した場合、その誤差に強く牽引され、一時的に大きな逸脱を生じるリスクがある。対照的に、SA は全データを等価に扱うため、特異な誤差を群衆全体で希釈し、平均的な安定性を保つことに長けている。

以上のことから、ヒット率を重視するなら WA が優れており、最大誤差を回避する安全性を重視するなら SA が優れているという、トレードオフの関係があると考えられる。

SA_MM と WA_MM を比較すると SA_MM は-7.2 ポイント WA_MM は-13.5 ポイントであり SA_MM が優勢となった。マップマッチングは強力な補正機能を持つため、入力となる予測位置（SA/WA）の多少の精度差は吸収されてしまい、最終的な結果には表れにくいためであると考えられる。

表 5.1　GPS の誤差における比較

Mode	MAE_m	XTE_Avg_m	XTE_Hit_pct	Error_Settings
SA	20.03	12.58	44.14	Total=10.6m
SA	22.34	14.77	23.42	Total=16.8m
SA_MM	19.89	8.96	61.26	Total=10.6m
SA_MM	20.69	10.65	54.05	Total=16.8m
WA	20.4	13.38	33.33	Physical=7.5m, GPS=7.5m
WA	23.21	14.5	28.83	Physical=7.5m, GPS=15m
WA_MM	20.38	8.98	64.86	Physical=7.5m, GPS=7,5m
WA_MM	22.59	11.72	51.35	Physical=7.5m, GPS=15.0m

5.4 実用性

図 5.9「MAE」,5.10「XTE」,5.11「XTE のヒット率」より、1〜6 台では、個々の GPS 誤差が相殺されずに直接反映されるため、MAE および XTE の値が大きい。特に 5 台未満では、XTE ヒット率が 50%を下回るケースも見られ、システムとしての信頼性が不十分であることが示唆された。

端末数が 7 台に達した時点で、全ての評価指標において劇的な精度の向上が確認された。

- MAE：21.21m→19.33m
- XTE：11m→8.27m
- XTE ヒット率 52.25%→66.67%

7 台を超えると精度の改善曲線は平坦化し、15 台に至るまで大きな変化は見られなかった。つまり、これ以上端末数を増やしても、コストに対する精度の向上効果は限定的であることがわかった。

以上の結果から、本システムが安定して機能するための最低要件は、緊急車両周辺に、アクティブな端末が 7 台以上存在することであると考察できる。

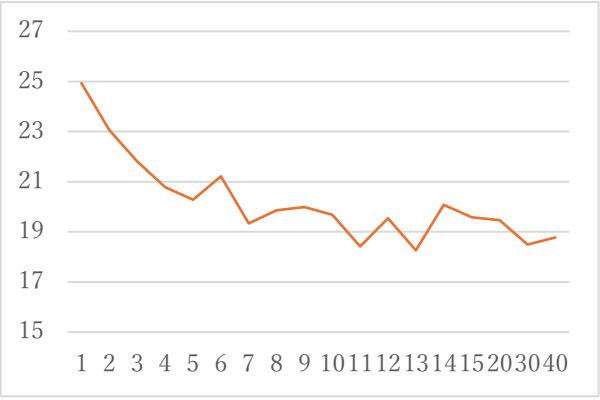

図 5.9　MAE

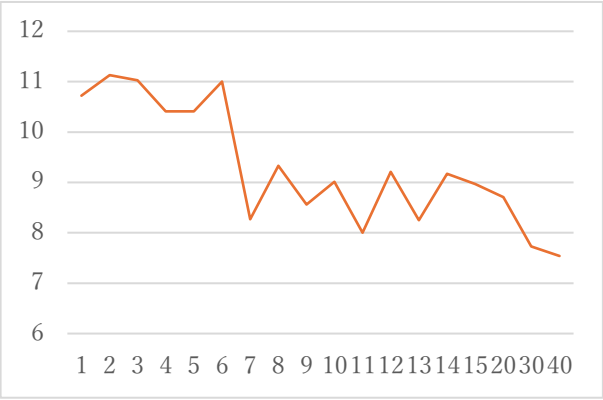

図 5.10　XTE

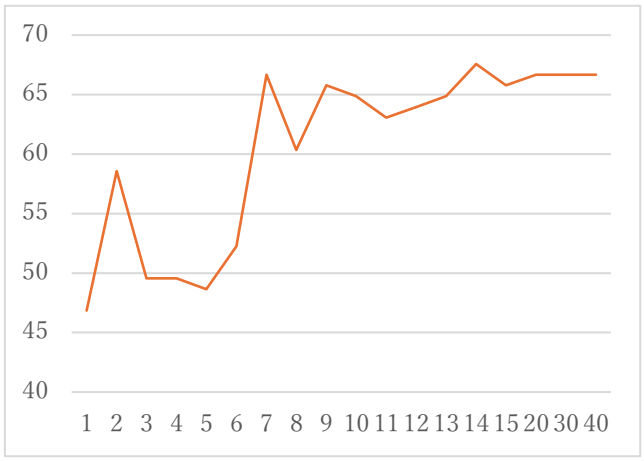

図 5.11　XTE のヒット率

5.5 考察

本節では、一連の評価実験（密度特性、追従性能、耐ノイズ性）の結果を統合し、本提案システムが社会実装され、実用的な緊急車両支援システムとして機能するための条件および適用環境について考察する。

5.5.1 端末密度

システムの実用化において最大の障壁となるのは、位置特定に必要なユーザー数（端末密度）の確保である。 密度特性の検証結果より、本システムの推定精度は端末数に対して非線形な向上を示し、特に 7 台 を境界として性能が劇的に安定することが確認された。

(1) 最低稼働要件

緊急車両の周囲半径 30m 以内に、常時 7 台以上のアクティブな端末が存在すること。この条件下であれば、マップマッチング後の XTE ヒット率は安定域に入り、誤った道路への吸着リスクを大幅に低減できる。都市部の交差点や主要幹線道路においては、信号待ちの車両ドライバーや歩行者を含めることで、この密度は十分に達成可能な水準であると考えられる。

(2) 推奨環境

より高精度な支援を行うためには、20 台程度の密度が望ましい。この密度帯域においては、WA が SA に対して明確な優位性を示し始め、安定した位置推定が可能となる。将来的には、車車間通信（V2V）等の普及により、物理的な人数に依存せず高頻度なデータ送信が可能となれば、この要件はさらに容易に満たされると考えられる。

5.5.2 交通環境

追従性能および耐ノイズ性の検証結果から、本システムが最も効果を発揮する「得意なシチュエーション」と、適用が難しい「限界領域」が明らかになった。

(1) 最適適用領域

都市部の交差点および渋滞区間 車両速度が 30km/h 以下 の低〜中速域において、本システムは極めて高い追従性能（ATE 数メートル以内）と道路判定精度（XTE ヒット率 80% 以上）を発揮した。これは、緊急車両にとって最も進路啓開支援が必要となる「交差点への進入時」や「渋滞通過時」、「信号待ち」の状況と合致しており、本システムが最も支援を必要とする場面で最大の性能を発揮できることを示唆している。

(2) 運用の限界

一方、60km/h を超える高速走行時には、移動平均処理に伴う予測位置のオーバーシュートが顕著となった。したがって、高速道路や郊外のバイパス等を高速で走行する場面では、予測精度の低下を考慮した運用が必要となる。

(3) ロバスト性の発揮

ビル街等の GPS 難視聴エリア GPS 誤差が増大する悪条件下においても、WA は SA と比較して精度の劣化を最小限に抑え、実用的な道路判定能力を維持した。これにより、高層ビルにより GPS マルチパスが発生しやすい都市中心部においても、システムが破綻することなく継続的な支援が可能であると結論付けられる。

6. まとめ

本研究では、緊急車両の接近を一般車両や歩行者に早期かつ確実に通知することを目的とし、スマートフォンから得られる位置情報と音量データを活用した位置推定システムを提案した。 提案手法では、各端末が観測したサイレン音量に応じて重み付けを行う加重平均法（WA）により位置推定を行い、さらに道路ネットワーク情報を用いたマップマッチング（MM）処理を適用することで、都市部における高精度な位置特定を目指した。
提案システムの有効性を検証するため、多様な交通状況や GPS 誤差環境を再現可能なシミュレータを構築し、計 70 パターン以上の条件下で定量的評価を行った。 実験結果より、以下の知見が得られた。
(1) 密度特性と実用化要件
提案手法は端末密度の増加に伴い精度が向上し、特に半径 30m 以内に 7 台以上の端末が存在する環境下において、実用的な安定性を示すことが明らかになった。また、20 台以上の密度においては、WA が SA を上回る推定精度を実現した。
(2) 追従性能と適用領域
車両速度の上昇に伴い、移動平均処理に起因する縦方向誤差（ATE）が増大するものの、30km/h 以下の低〜中速域においては極めて高い追従性能（ATE 数メートル以内、道路内収束率 80%以上）を発揮した。これにより、本システムは緊急車両の支援が最も必要とされる「交差点進入時」や「渋滞通過時」に最適化された特性を持つと考えられる。
(3) 耐ノイズ性とロバスト性
GPS 測位環境が悪化する都市部を想定した実験において、SA が精度の崩壊（ヒット率 20%台）を起こす中、WA は音量情報によるフィルタリング効果により、XTE ヒット率の劣化を最小限に抑える高いロバスト性を示した。
以上の結果より、位置情報と音量データを組み合わせた本提案手法は、都市部の交通環境における緊急車両の走行支援を実現するための有望なアプローチであると結論付ける。

6.1 今後の課題

本研究では、位置予測に等速直線運動モデルを、距離推定に自由空間での減衰モデルを採用し、システムを構築した。しかし、実環境においては、カーブ区間での予測位置の逸脱（オーバーシュート）や、建物による遮蔽・環境音の影響による音量変動といった、単純なモデルでは捉えきれない事象が発生し得る。
したがって今後の課題として、道路形状や車両挙動を考慮した予測アルゴリズム（カルマンフィルタやマップマッチング連動型予測など）の導入と、実環境で収集した音響データに基づくロバストな距離推定モデル（機械学習や周波数解析など）の構築を並行して進め、複雑な都市部環境における推定精度と信頼性をさらに向上させる必要がある。

また、本実験では、全ての端末が同一の条件で音量を観測できる理想環境を想定した。しかし、実社会への実装においては、車両の遮音性の違いや、端末ごとのマイク性能のばらつきが大きな課題となる。単純な絶対音量の比較では、遮音性の高い車内にいる端末が「遠くにいる」と誤判定されるリスクがある。

この課題に対し、今後は音量の絶対値のみに依存せず、各端末における「時系列変化」に着目した検知ロジックの導入が必要である。接近に伴う音量増加のトレンドを解析することで、端末や環境ごとの感度差を相殺し、よりロバストな位置推定が可能になると考えられる。

謝辞

本研究を進めるにあたり、研究や研究以外についても様々な助言、指導をして下さった三好教授に心から御礼申し上げます。
また、研究の助言や議論をして下さり、また、日常生活でもお世話になった、三好研究室4回生の皆様および大学院生の方々に、心から御礼申し上げます

参考文献

[1] 道路交通法（昭和 35 年法律第 105 号）

[2] 一般財団法人 救急振興財団. "救急車による交通事故及び 事故防止に関する取り組みの全国調査". (入手先: https://fasd.jp/files/libs/805/202201311345501064.pdf, 2025 年 10 月 28 日アクセス).

[3] 佐藤公治. "運転初心者と熟達者の視覚探索 周辺視情報処理". *IATSS REVIEW (国際交通安全学会誌)*, Vol. 19, No. 3, pp. 216-222, 1993. (入手先: https://www.iatss.or.jp/entry_img/19-3-07.pdf, 2025 年 10 月 28 日アクセス).

[4] ジェネクスト株式会社. "運転初心者・若年ドライバーに多い交通事故 3 選". (出典：公益財団法人交通事故総合分析センター 交通事故統計データ). (入手先: https://kantei.genext.co.jp/youth/, 2025 年 10 月 28 日アクセス).

[5] 総務省. "人命救助等におけるＧＰＳ位置情報の取扱いに 関するとりまとめ". (入手先: https://www.soumu.go.jp/main_content/000237319.pdf, 2025 年 10 月 28 日アクセス).

[6] トヨタ自動車株式会社. "ITS Connect". (入手先: https://toyota.jp/technology/safety/itsconnect/, 2025 年 10 月 29 日アクセス).

[7] トヨタ自動車株式会社. "Background of the 'ITS Connect'". (入手先: https://www.toyota.co.jp/pages/contents/jpn/tech/its/world_congress/2016melbourne/pdf/Background_of_the_ITS_Connect.pdf, 2025 年 10 月 29 日アクセス).

[8] ITS Connect 推進協議会. "後付け車載機開発で車の安全性を高める ITS Connect の一層の推進へ". 2023 年 11 月 15 日. (入手先: https://www.itsconnect-pc.org/wp/wp-content/uploads/2023/11/20231115_PR.pdf, 2025 年 10 月 29 日アクセス).

[9] 国立研究開発法人情報通信研究機構. "救急車の位置情報をサイレン音に載せて周囲のカーナビに表示". (入手先: https://www.nict.go.jp/out-promotion/technology-transfer/partner/watermark.html, 2025 年 10 月 29 日アクセス).

付録

main_map.py(サーバー、マップマッチング)

```
# main.py (音量対応版)

import sqlite3
from fastapi import FastAPI
from pydantic import BaseModel
from datetime import datetime
from predictor import predict_next_location # (これは次のステップで修正します)

# --- ▼ マップマッチングに必須 ▼ ---
import osmnx as ox
import networkx as nx
from shapely.geometry import Point, LineString
from shapely.ops import nearest_points
# --- ▲ マップマッチングに必須 ▲ ---

# --- データベースの準備 ---
DB_NAME = 'locations.db'

def setup_database():
    conn = sqlite3.connect(DB_NAME)
    cursor = conn.cursor()

    # detections テーブルに sound_intensity カラムを追加
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS detections (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            latitude REAL NOT NULL,
            longitude REAL NOT NULL,
            sound_intensity REAL NOT NULL,
            timestamp TEXT NOT NULL
        )
    ''')

    # predictions テーブル (変更なし)
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS predictions (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            latitude REAL,
            longitude REAL,
            timestamp TEXT NOT NULL
        )
    ''')

    conn.commit()
    conn.close()
    print(f"データベース '{DB_NAME}' の準備が完了しました。")

# --- FastAPI アプリの準備 ---

# スマホから受け取るデータに sound_intensity を追加
class LocationData(BaseModel):
    latitude: float
    longitude: float
    sound_intensity: float

app = FastAPI()

# --- ▼ サーバー起動時に 1 回だけ地図をロード ▼ ---
print("高松市の地図データをロードしています...（数秒～数十秒かかります）")
place_name = "Takamatsu, Kagawa, Japan"
try:
    # G をグローバル変数としてロードし、全関数で使えるようにする
    G = ox.graph_from_place(place_name, network_type='drive')
    print("地図データのロード完了。")
except Exception as e:
    print(f"エラー: 地図データのダウンロードに失敗しました。{e}")
    G = None # エラーの場合は G を None にしておく
# --- ▲ ここまで ▲ ---

@app.get("/")
def read_root():
    return {"message": "緊急車両通知支援システム API サーバー"}

@app.post("/location")
def receive_location(data: LocationData):
    now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    try:
        conn = sqlite3.connect(DB_NAME)
        cursor = conn.cursor()
        # 保存するデータに音量を追加
        cursor.execute(
            "INSERT INTO detections (latitude, longitude, sound_intensity, timestamp) VALUES (?, ?, ?, ?)",
            (data.latitude, data.longitude, data.sound_intensity, now)
        )
        conn.commit()
        conn.close()
    except Exception as e:
        print(f"❌ [保存失敗] エラー: {e}")
    return {"status": "ok"}


@app.get("/predict")
def get_prediction():
    # 1. 従来通りの「仮の予測」を計算
    # (predictor.py は空の場合 {"lat": None, "lon": None} を返すようになっています)
    predicted_location = predict_next_location()

    if predicted_location:
        try:
            # --- ★追加変更: データが空 (NULL) かどうかをチェック ---
            # 空データ (None) の場合は、マップマッチングをスキップしてそのまま保存へ進む
            if predicted_location['lat'] is None:
                final_pred_lat = None
                final_pred_lon = None
                mm_status = "SKIPPED (No Data)"

            else:
```

```
        # --- ▼ 2. マップマッチングによる補正処理（データがある時だけ実行） ▼ ---

        # G がロード失敗していたらエラー
        if G is None:
            return {"status": "error", "message": "Map data (G) is not loaded."}

        # 2a. 仮の予測位置を取得
        raw_lat = predicted_location['lat']
        raw_lon = predicted_location['lon']

        # 2b. 仮の予測位置を「点」オブジェクトに変換 (経度, 緯度の順)
        raw_point_geom = Point(raw_lon, raw_lat)

        # 2c. 地図(G)から「最も近い道路」を高速検索 (経度, 緯度の順)
        edge_id = ox.nearest_edges(G, X=raw_lon, Y=raw_lat)

        # 2d. 道路の形状データ(LineString)を取得
        edge_data = G.edges[edge_id]
        if 'geometry' in edge_data:
            road_geom = edge_data['geometry']
        else:
            u_node = G.nodes[edge_id[0]]
            v_node = G.nodes[edge_id[1]]
            road_geom = LineString([(u_node['x'], u_node['y']), (v_node['x'], v_node['y'])])

        # 2e. 「仮の予測点」を「道路」の上にスナップ（引き戻し）
        snapped_point_on_road, _ = nearest_points(road_geom, raw_point_geom)

        # 2f. 補正された「最終的な予測位置」を取得
        final_pred_lon = snapped_point_on_road.x
        final_pred_lat = snapped_point_on_road.y
        mm_status = "DONE"

        # --- ▲ マップマッチング処理ここまで ▲ ---
    # --- 3. 最終的な予測結果（または NULL）を DB に保存 ---
    now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    conn = sqlite3.connect(DB_NAME)
    cursor = conn.cursor()

    # final_pred_lat が None なら、DB には NULL として保存される
    cursor.execute(
        "INSERT INTO predictions (latitude, longitude, timestamp) VALUES (?, ?, ?)",
        (final_pred_lat, final_pred_lon, now)
    )
    conn.commit()
    conn.close()

    # ログ出力
    if final_pred_lat is not None:
        print(f"補正後の予測位置: 緯度={final_pred_lat:.6f}, 経度={final_pred_lon:.6f} ({mm_status})")
    else:
        print(f"予測: データ不足のため空データを記録しました ({mm_status})")

    # 返り値も補正後のものにする
    final_output = {"lat": final_pred_lat, "lon": final_pred_lon}
    return {"status": "success", "predicted_location": final_output}

  except Exception as e:
    print(f"予測またはマップマッチング中にエラー: {e}")
    return {"status": "error", "message": f"Prediction failed: {e}"}

else:
  return {"status": "error", "message": "Not enough data to predict."}

@app.get("/latest_prediction")
def get_latest_prediction():
  """ データベースから最新の予測結果を 1 件取得して返す """

  print("DEBUG: /latest_prediction が呼び出されました。") # ★呼び出し確認ログ

  # --- 予測計算の実行 ---
  predicted_location = predict_next_location()

  # --- 予測結果の DB 保存 ---
  if predicted_location:
    try:
      now = datetime.now().strftime('%Y-m-%d %H:%M:%S')
      conn_save = sqlite3.connect(DB_NAME) # 保存用コネクション
      cursor_save = conn_save.cursor()
      print(f"DEBUG: predictions テーブルへの保存試行: lat={predicted_location['lat']}, lon={predicted_location['lon']}, time={now}") # ★保存試行ログ
      cursor_save.execute(
        "INSERT INTO predictions (latitude, longitude, timestamp) VALUES (?, ?, ?)",
        (predicted_location['lat'], predicted_location['lon'], now)
      )
      conn_save.commit()
      conn_save.close()
      print("DEBUG: predictions テーブルへの保存成功。") # ★保存成功ログ
    except Exception as e:
      # ★★★★★ エラー発生時に、具体的なエラー内容を出力 ★★★★★
```

```
        print(f"❌ ERROR: 予測結果の DB 保存中にエラーが発生しました: {e}")
        print(f"エラー発生時のデータ: {predicted_location}")
        # ★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★
        # 保存に失敗しても、後続の取得処理は試みる
        pass # エラーを握りつぶさず、ログだけ出して次に進む

    # --- DB から最新の予測結果を取得 ---
    try:
        conn_fetch = sqlite3.connect(DB_NAME) # 取得用コネクション
        cursor_fetch = conn_fetch.cursor()
        cursor_fetch.execute("SELECT latitude, longitude FROM predictions ORDER BY id DESC LIMIT 1")
        result = cursor_fetch.fetchone()
        conn_fetch.close()

        if result:
            print(f"DEBUG: 最新予測の取得成功: lat={result[0]}, lon={result[1]}") # ★取得成功ログ
            return {"status": "success", "latitude": result[0], "longitude": result[1]}
        else:
            print("DEBUG: predictions テーブルに有効なデータがありません。") # ★データなしログ
            return {"status": "nodata", "message": "予測データがまだありません。"}

    except Exception as e:
        print(f"❌ ERROR: 最新予測の DB 取得中にエラーが発生しました: {e}")
        return {"status": "error", "message": "予測の取得に失敗しました。"}

# サーバー起動時にデータベースの準備をする
setup_database()
```

predictor.py(予測モデル)

```
# predictor.py (加重平均モデル)

import sqlite3
import pandas as pd

# --- 設定 ---
DB_NAME = 'locations.db'
TIME_WINDOW_NOW = 2  # 「現在」と見なす時間窓（秒）
TIME_WINDOW_PAST = 5 # 「過去」と比較するための時間窓（秒）

def predict_next_location():
    """
    データベースのデータから、音の強弱を重みとした加重平均を用いて、
    緊急車両の次の位置を予測する関数。
    """
    try:
        conn = sqlite3.connect(DB_NAME)

        # --- STEP 1: 現在位置の「推定」 (加重平均を使用) ---

        # データベースから、音量データも含めて最新のデータ群を取得する
        query_now = f"SELECT latitude, longitude, sound_intensity FROM detections WHERE timestamp >= datetime('now', '-{TIME_WINDOW_NOW} seconds', 'localtime')"
        df_now = pd.read_sql_query(query_now, conn)


        # ▼▼▼ デバッグログ追加 ▼▼▼
        print(f"DEBUG: 現在データ取得結果: {len(df_now)}件")

        # データが十分にない、または音量がゼロの場合は予測不可能
        #一時的に 5 に変更
        #if len(df_now) < 5: #or df_now['sound_intensity'].sum() == 0:
            #print("予測モデル: 現在データ不足または音量ゼロのためスキップ") # ログ変更
            #conn.close()
            #return None
            # ★変更点: None ではなく、明示的に「空データ」を返す
            #return {"lat": None, "lon": None}

        # 1. 件数チェック
        if len(df_now) < 1:
            print("予測モデル: 現在データ不足のため空データを返します")
            conn.close() # ★必ず閉じる
            return {"lat": None, "lon": None} # ★空データを返す

        # 2. 音量合計（分母）のチェック ★ここが重要
        sound_sum_now = df_now['sound_intensity'].sum()
        if sound_sum_now == 0:
            print("予測モデル: 現在データの音量合計が 0 のため空データを返します")
            conn.close() # ★必ず閉じる
            return {"lat": None, "lon": None} # ★空データを返す



        # 取得したデータ群の「加重平均」を計算し、「現在の中心点」とする
        # (緯度 * 音量)の合計を、音量の合計で割る
```

```
    #current_center_lat = (df_now['latitude'] *
df_now['sound_intensity']).sum() /
df_now['sound_intensity'].sum()
    #current_center_lon = (df_now['longitude'] *
df_now['sound_intensity']).sum() /
df_now['sound_intensity'].sum()
    current_center_lat = (df_now['latitude'] *
df_now['sound_intensity']).sum() / sound_sum_now
    current_center_lon = (df_now['longitude'] *
df_now['sound_intensity']).sum() / sound_sum_now
    current_center = {
      "lat": current_center_lat,
      "lon": current_center_lon
    }

    # --- STEP 2: 未来位置の「予測」 ---

    # 比較対象となる、少し前のデータ群を音量データ
も含めて取得する
    query_past = f"SELECT latitude, longitude,
sound_intensity FROM detections WHERE timestamp
BETWEEN datetime('now', '-{TIME_WINDOW_NOW +
TIME_WINDOW_PAST} seconds', 'localtime') AND
datetime('now', '-{TIME_WINDOW_PAST} seconds',
'localtime')"
    df_past = pd.read_sql_query(query_past, conn)


    # ▼▼▼ デバッグログ追加 ▼▼▼
    print(f"DEBUG: 過去データ取得結果: {len(df_past)}
件")

    # 過去のデータも十分にない、または音量がゼロの
場合は予測不可能
    #一時的に 5 に変更
    #if len(df_past) < 5: #or
df_past['sound_intensity'].sum() == 0:
      #print("予測モデル: 過去データ不足または音量ゼ
ロのためスキップ") # ログ変更
      #conn.close()
      #return None
      # ★変更点: None ではなく、明示的に「空デー
タ」を返す
      #return {"lat": None, "lon": None}

    # 3. 過去データの件数チェック
    if len(df_past) < 1:
      print("予測モデル: 過去データ不足のため空データ
を返します")
      conn.close() # ★必ず閉じる
      return {"lat": None, "lon": None} # ★空データを返す

    # 4. 過去データの音量合計チェック ★ここも重要
    sound_sum_past = df_past['sound_intensity'].sum()
    if sound_sum_past == 0:
      print("予測モデル: 過去データの音量合計が 0 のた
め空データを返します")
      conn.close() # ★必ず閉じる
      return {"lat": None, "lon": None} # ★空データを返す

    # 同様に「過去の中心点」も加重平均で計算
    #past_center_lat = (df_past['latitude'] *
df_past['sound_intensity']).sum() /
df_past['sound_intensity'].sum()
    #past_center_lon = (df_past['longitude'] *
df_past['sound_intensity']).sum() /
df_past['sound_intensity'].sum()
    past_center_lat = (df_past['latitude'] *
df_past['sound_intensity']).sum() / sound_sum_past
    past_center_lon = (df_past['longitude'] *
df_past['sound_intensity']).sum() / sound_sum_past

    past_center = {
      "lat": past_center_lat,
      "lon": past_center_lon
    }

    # 2 つの中心点から、動き（ベクトル）を計算 (変更
なし)
    velocity_vector = {
      "lat": current_center["lat"] - past_center["lat"],
      "lon": current_center["lon"] - past_center["lon"]
    }

    # 未来の位置を予測 (変更なし)
    predicted_location = {
      "lat": current_center["lat"] + velocity_vector["lat"],
      "lon": current_center["lon"] + velocity_vector["lon"]
    }

    # デバッグ用に計算結果をターミナルに表示
    print("\n--- 予測計算実行 (加重平均) ---")
    print(f"現在中心: {current_center['lat']:.4f},
{current_center['lon']:.4f} ({len(df_now)}点から推定)")
    print(f"過去中心: {past_center['lat']:.4f},
{past_center['lon']:.4f} ({len(df_past)}点から推定)")
    print(f"予測位置: {predicted_location['lat']:.4f},
{predicted_location['lon']:.4f}")
    print("--------------------")

    conn.close()
    return predicted_location

  except Exception as e:
    print(f"予測モデルでエラーが発生しました: {e}")
    return None
```

evaluate.py(評価スクリプト)

```
import sqlite3
import pandas as pd
import math
import numpy as np
from shapely.geometry import Point, LineString
import argparse
import os

# --- 引数設定 ---
parser = argparse.ArgumentParser()
parser.add_argument('--speed', type=int, required=True,
help='実験速度(km/h)')
```

```
parser.add_argument('--phones', type=int, required=True, help='スマホ台数')
parser.add_argument('--mode', type=str, required=True, help='実験モード名')
parser.add_argument('--output_detail', type=str, required=True, help='詳細データの保存先 CSV')
parser.add_argument('--append_summary', type=str, required=True, help='まとめ結果を追記する CSV')
parser.add_argument('--exp_id', type=str, default="00", help='実験番号')
parser.add_argument('--sigma_info', type=str, default="", help='誤差設定')

args = parser.parse_args()

# --- 設定 ---
DB_NAME = 'locations.db'
GROUND_TRUTH_FILE = 'ground_truth.csv'

# --- 論理パラメータ ---
PREDICT_LEAD_TIME = 4     # 4 秒先を予測
IGNORE_INITIAL_STEPS = 7   # 立ち上がり 0~6 秒は無視

# 縦方向の許容誤差 (速度から自動計算)
ATE_THRESHOLD_METERS = (args.speed / 3.6) * 4.0

# --- 座標変換用定数 ---
LAT_TO_M = 110946.0
LON_TO_M = 91165.0

def to_xy(lat, lon):
  y = lat * LAT_TO_M
  x = lon * LON_TO_M
  return x, y

def calculate_haversine_distance(lat1, lon1, lat2, lon2):
  R = 6371e3
  phi1 = math.radians(lat1); phi2 = math.radians(lat2)
  delta_phi = math.radians(lat2 - lat1)
  delta_lambda = math.radians(lon2 - lon1)
  a = math.sin(delta_phi / 2)**2 + math.cos(phi1) * math.cos(phi2) * math.sin(delta_lambda / 2)**2
  c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
  return R * c

def evaluate_model():
  print(f"--- 評価実行 [ID:{args.exp_id}] ---")

  # 1. データ読み込み
  try:
    if not os.path.exists(DB_NAME):
      print(f"エラー: データベース {DB_NAME} が見つかりません。")
      return

    df_truth = pd.read_csv(GROUND_TRUTH_FILE)
    conn = sqlite3.connect(DB_NAME)
    df_pred = pd.read_sql_query("SELECT latitude, longitude, timestamp FROM predictions ORDER BY id", conn)
    conn.close()
  except Exception as e:
    print(f"エラー: データの読み込みに失敗しました: {e}")
    return

  if len(df_pred) == 0:
    print("エラー: 予測データが 0 件です。")
    return

  # 2. 正解ルート線作成
  truth_points_xy = [to_xy(r['correct_lat'], r['correct_lon']) for _, r in df_truth.iterrows()]
  truth_line = LineString(truth_points_xy)

  results = []
  list_xte = []
  list_ate = []
  list_euclid = []

  # 3. 評価ループ
  for t in range(len(df_pred)):
    if t < IGNORE_INITIAL_STEPS: continue
    pred_row = df_pred.iloc[t]
    if pred_row['latitude'] is None: continue

    target_time = t + PREDICT_LEAD_TIME
    if target_time >= len(df_truth): break

    truth_row = df_truth.iloc[target_time]

    # 計算
    pred_x, pred_y = to_xy(pred_row['latitude'], pred_row['longitude'])
    pred_point = Point(pred_x, pred_y)

    truth_x, truth_y = to_xy(truth_row['correct_lat'], truth_row['correct_lon'])
    truth_point = Point(truth_x, truth_y)

    xte = truth_line.distance(pred_point)

    dist_pred = truth_line.project(pred_point)
    dist_truth = truth_line.project(truth_point)
    ate = dist_truth - dist_pred

    euclid = calculate_haversine_distance(truth_row['correct_lat'], truth_row['correct_lon'], pred_row['latitude'], pred_row['longitude'])

    list_xte.append(xte)
    list_ate.append(ate)
    list_euclid.append(euclid)

    results.append({
      "Sim_Step": t, "Truth_Time": target_time,
      "Pred_Lat": pred_row['latitude'], "Pred_Lon": pred_row['longitude'],
      "Truth_Lat": truth_row['correct_lat'], "Truth_Lon": truth_row['correct_lon'],
      "Error_Euclid_m": euclid, "Error_XTE_m": xte, "Error_ATE_m": ate
    })

  if len(results) == 0:
    print("有効データがありませんでした。")
    return
```

```
    # 集計
    mae = np.mean(list_euclid)
    rmse = np.sqrt(np.mean(np.array(list_euclid)**2))
    avg_xte = np.mean(list_xte)
    avg_ate = np.mean(np.abs(list_ate))
    xte_hit = (np.sum(np.abs(list_xte) <= 5.0) /
len(list_xte)) * 100
    ate_hit = (np.sum(np.abs(list_ate) <=
ATE_THRESHOLD_METERS) / len(list_ate)) * 100

    # ★検知データ数の計算 (スマホ台数 × 有効予測回数)
    detection_count = args.phones * len(results)

    print(f"  -> 結果 [ID:{args.exp_id}]: MAE={mae:.2f}m, 検
知数={detection_count}")

    # [A] 詳細データの保存
    pd.DataFrame(results).to_csv(args.output_detail,
index=False)

    # ★★★ ここで誤差情報を追記 ★★★
    if args.sigma_info:
        with open(args.output_detail, 'a') as f:
            # CSV の構造を壊さないよう、コメント行(#)とし
て、または新しい行として追加
            f.write(f"\n# Error_Settings,{args.sigma_info}")

    print(f"  -> 詳細保存: {args.output_detail} (設定値追記
済み)")

    # [B] まとめファイルへの追記
    summary_data = {
        "Exp_ID": args.exp_id,
        "Mode": args.mode,
        "Speed_kmh": args.speed,
        "Phones": args.phones,
        "MAE_m": round(mae, 2),
        "RMSE_m": round(rmse, 2),
        "XTE_Avg_m": round(avg_xte, 2),
        "ATE_Avg_m": round(avg_ate, 2),
        "XTE_Hit_pct": round(xte_hit, 2),
        "ATE_Hit_pct": round(ate_hit, 2),
        "ATE_Threshold_m":
round(ATE_THRESHOLD_METERS, 2),
        "Valid_Count": len(results),
        "Detection_Count": detection_count,
        "Error_Settings": args.sigma_info # ★まとめ CSV にも
記録
    }

    df_summary = pd.DataFrame([summary_data])

    if not os.path.isfile(args.append_summary):
        df_summary.to_csv(args.append_summary,
index=False, encoding='utf-8-sig')
    else:
        df_summary.to_csv(args.append_summary, mode='a',
header=False, index=False, encoding='utf-8-sig')

    print(f"  -> まとめ追記完了")

if __name__ == "__main__":
    evaluate_model()
```

simulator.py(シミュレーター)

```
# simulator.py (加重平均)

import requests
import time
import random
import osmnx as ox
import networkx as nx
import math
import csv
import argparse  # ★追加

# --- ▼▼▼ 引数設定（ここを書き換える） ▼▼▼ ---
parser = argparse.ArgumentParser()
parser.add_argument('--phones', type=int, default=20,
help='スマホ台数')
parser.add_argument('--speed', type=int, default=40,
help='速度(km/h)')
args = parser.parse_args()

# --- 設定項目 ---
SERVER_URL = "http://127.0.0.1:8003/location"
NUMBER_OF_PHONES = args.phones  # ★引数を使う
VEHICLE_SPEED_KMPH = args.speed # ★引数を使う
GROUND_TRUTH_FILE = 'ground_truth.csv'

# --- ▼▼▼ 正規分布（WA 用：2 種類の誤差） ▼▼▼ ---

# 1. 誤差 A: スマホの「物理的な」散らばり (σ_physical)
PHYSICAL_SIGMA_METERS = 7.5  # ★新規追加 (この値は
仮定)

# 2. 誤差 B: スマホの「GPS 測定」誤差 (σ_gps)
GPS_SIGMA_METERS = 15    # ★これが 7.5m

# 3. 高松市の緯度（シミュレーションの中心）
TAKAMATSU_LATITUDE = 34.34

# 4. メートルを「度」に変換するための定数を計算
METERS_PER_DEGREE_LAT = 111000.0
METERS_PER_DEGREE_LON = 111000.0 *
math.cos(math.radians(TAKAMATSU_LATITUDE))

# 5.「度」単位の標準偏差 (sigma) を計算
# 物理的な散らばり用 (σ=15m)
PHYSICAL_SIGMA_LAT_DEG = PHYSICAL_SIGMA_METERS
/ METERS_PER_DEGREE_LAT
PHYSICAL_SIGMA_LON_DEG = PHYSICAL_SIGMA_METERS
/ METERS_PER_DEGREE_LON
# GPS 測定誤差用 (σ=7.5m)
GPS_SIGMA_LAT_DEG = GPS_SIGMA_METERS /
METERS_PER_DEGREE_LAT
GPS_SIGMA_LON_DEG = GPS_SIGMA_METERS /
METERS_PER_DEGREE_LON

print(f"WA シミュレータ (最終モデル) 起動")
print(f" 物理的散らばり:
σ={PHYSICAL_SIGMA_METERS}m")
print(f" GPS 測定誤差: σ={GPS_SIGMA_METERS}m")
```

```
print(f" (SA 側は σ={math.sqrt(7.5**2 + 15**2):.2f}m で実行してください)")
# --- ▲▲▲ ここまでが新しい設定 ▲▲▲ ---


# --- ユーティリティ関数 ---
# ... (calculate_haversine_distance は変更なし) ...
def calculate_haversine_distance(lat1, lon1, lat2, lon2):
  R = 6371e3
  phi1 = math.radians(lat1); phi2 = math.radians(lat2)
  delta_phi = math.radians(lat2 - lat1)
  delta_lambda = math.radians(lon2 - lon1)
  a = math.sin(delta_phi / 2)**2 + math.cos(phi1) * math.cos(phi2) * math.sin(delta_lambda / 2)**2
  c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
  return R * c

# --- 1. 地図データのダウンロードとルート計算 ---
# ... (変更なし) ...
print("高松市の道路網データをダウンロードしています...")
place_name = "Takamatsu, Kagawa, Japan"
G = ox.graph_from_place(place_name, network_type='drive')

start_latlon = (34.335016,134.051001) # ある地点 A
end_latlon = (34.341181,134.043657)   # 赤十字病院
#start_latlon = (34.347641,134.059944) # 直進スタート
#end_latlon = (34.347044,134.069018)   # 直進ゴール
#start_latlon = (34.339827,134.041924) # 一方通行
#end_latlon = (34.339466,134.044353)   # 一方通行

start_node = ox.nearest_nodes(G, start_latlon[1], start_latlon[0])
end_node = ox.nearest_nodes(G, end_latlon[1], end_latlon[0])
print("最短経路を計算しています...")
route_nodes = nx.shortest_path(G, start_node, end_node, weight='length')
route_length_meters = nx.shortest_path_length(G, start_node, end_node, weight='length')
speed_mps = VEHICLE_SPEED_KMPH * 1000 / 3600
edge_lengths = ox.routing.route_to_gdf(G, route_nodes)['length'].tolist()
print(f"ルートの総距離: {route_length_meters:.0f} m")
print(f"経由する交差点の数: {len(route_nodes)}ヶ所")

# --- 2. シミュレーション本体 ---
print(f"\n シミュレーションを開始します... (正解データを {GROUND_TRUTH_FILE} に保存)")

with open(GROUND_TRUTH_FILE, 'w', newline='') as f:
    # ... (CSV ライターの準備、ヘッダー書き込み) ...
    writer = csv.writer(f)
    writer.writerow(['time_step', 'correct_lat', 'correct_lon'])

    for i in range(len(route_nodes) - 1):
        # ... (セグメント処理、正解位置(correct_lat/lon)の計算) ...
        u, v = route_nodes[i], route_nodes[i+1]
        segment_length = edge_lengths[i]
        travel_time_for_segment = segment_length / speed_mps if speed_mps > 0 else 1
        num_steps_in_segment = round(travel_time_for_segment) if travel_time_for_segment > 0 else 1
        start_point = G.nodes[u]
        end_point = G.nodes[v]
        for t_segment in range(num_steps_in_segment):
          progress = t_segment / travel_time_for_segment if travel_time_for_segment > 0 else 1.0
          correct_lat = start_point['y'] + (end_point['y'] - start_point['y']) * progress
          correct_lon = start_point['x'] + (end_point['x'] - start_point['x']) * progress
          total_time_step = sum([round(l / speed_mps) if speed_mps > 0 else 1 for l in edge_lengths[:i]]) + t_segment
          print(f"\n--- 時刻: {total_time_step}秒 ---")
          print(f"正解位置: 緯度={correct_lat:.4f}, 経度={correct_lon:.4f}")
          writer.writerow([total_time_step , correct_lat, correct_lon])

          # 仮想スマホからのデータ送信
          for _ in range(NUMBER_OF_PHONES):

            # --- ▼▼▼ WA 用 最終モデルロジック ▼▼▼ ---

            # 1. 誤差 A: スマホの「真の」位置を決定 (σ=15m)
            true_pos_error_lat = random.normalvariate(0, PHYSICAL_SIGMA_LAT_DEG)
            true_pos_error_lon = random.normalvariate(0, PHYSICAL_SIGMA_LON_DEG)

            phone_true_lat = correct_lat + true_pos_error_lat
            phone_true_lon = correct_lon + true_pos_error_lon

            # 2. 「真の」音データを計算 (音の誤差は「なし」)
            true_distance = calculate_haversine_distance(correct_lat, correct_lon, phone_true_lat, phone_true_lon)
            sound_intensity = 1.0 / (true_distance + 1.0) # ★「真の」音

            # 3. 誤差 B: 「GPS 測定」誤差を生成 (σ=7.5m)
            gps_error_lat = random.normalvariate(0, GPS_SIGMA_LAT_DEG)
            gps_error_lon = random.normalvariate(0, GPS_SIGMA_LON_DEG)

            # 4. 「報告」位置を計算 = (真の位置 + GPS 誤差)
            phone_reported_lat = phone_true_lat + gps_error_lat
            phone_reported_lon = phone_true_lon + gps_error_lon
```

```
            # 5. 送信するデータ（「誤差ありの位置」と「真の音」）
            simulated_data = {
              "latitude": phone_reported_lat,  # ★ 誤差ありの位置
              "longitude": phone_reported_lon, # ★ 誤差ありの位置
              "sound_intensity": sound_intensity  # ★ 真の（完璧な）音
            }
            # --- ▲▲▲ ロジック変更ここまで ▲▲▲ ---

            try:
              requests.post(SERVER_URL, json=simulated_data, timeout=0.5)
            except requests.exceptions.RequestException: pass

          try:
            # サーバーの応答は待たない
            requests.get(SERVER_URL.replace("/location", "/predict"), timeout=0.1)
          except requests.exceptions.ReadTimeout: pass
          except requests.exceptions.RequestException: pass

          time.sleep(1) # ★安定のため 1 秒 に戻す

print("\n 目的地に到達しました。")
print(f"正解データを {GROUND_TRUTH_FILE} に保存しました。")
print("シミュレーションが完了しました。")
```

visualize.py(可視化スクリプト)

```
# visualize.py (完全版：正解ルートと音量グラデーション付き)

import folium
import pandas as pd
import sqlite3
import osmnx as ox # 地図データとルート描画のため再度インポート
import networkx as nx
import matplotlib.cm as cm # 色のグラデーションのためにインポート
import matplotlib.colors as mcolors # 色のグラデーションのためにインポート
import argparse # ★追加

# --- ▼▼▼ 引数設定 ▼▼▼ ---
parser = argparse.ArgumentParser()
parser.add_argument('--output', type=str, default='visualization.html', help='出力ファイル名')
args = parser.parse_args()


print("データベースからデータを読み込んでいます...")
DB_NAME = 'locations.db'
MAP_FILE_NAME = args.output # ★引数から受け取った名前を使う

try:
  conn = sqlite3.connect(DB_NAME)
  # sound_intensity も含めて読み込む
  df_detections = pd.read_sql_query("SELECT latitude, longitude, sound_intensity FROM detections", conn)
  df_predictions = pd.read_sql_query("SELECT latitude, longitude FROM predictions", conn)
  conn.close()

  # --- ▼▼▼ 修正ポイント：空データの削除 ▼▼▼ ---
  # NULL (None/NaN) が含まれている行を削除する
  initial_len_pred = len(df_predictions)
  df_predictions = df_predictions.dropna(subset=['latitude', 'longitude'])
  dropped_count = initial_len_pred - len(df_predictions)

  if dropped_count > 0:
    print(f"通知: 予測データから空のデータ（スキップ分）を {dropped_count} 件除外しました。")
  # --- ▲▲▲ 修正ここまで ▲▲▲ ---


  print(f"検知データ: {len(df_detections)}件, 予測データ: {len(df_predictions)}件 を読み込みました。")
except Exception as e:
  print(f"エラー: データベースの読み込みに失敗しました。 {e}")
  exit()

# --- 地図データと正解ルートの取得（simulator.py と同じロジック） ---
print("正解ルートを計算しています...")
place_name = "Takamatsu, Kagawa, Japan"
G = ox.graph_from_place(place_name, network_type='drive')



start_latlon = (34.335016,134.051001) # ある地点 A
end_latlon = (34.341181,134.043657)  # 赤十字病院

#start_latlon = (34.347641,134.059944) # 直進スタート
#end_latlon = (34.347044,134.069018)  # 直進ゴール
#start_latlon = (34.339827,134.041924) # 一方通行
#end_latlon = (34.339466,134.044353)  # 一方通行

#start_latlon = (34.343688, 134.041617) # 高松工芸
#start_latlon = (34.329261, 134.045442) # 栗林公園スタート
#start_latlon = (34.342368,134.073443) # 総合体育館
#start_latlon = (34.348429,134.047003) # 北警察
#end_latlon = (34.348494,134.063067)  # 中央病院
#end_latlon = (34.340687,134.044412)  # 赤十字病院
#end_latlon = (34.329261, 134.045442)  # 栗林公園
```

```
start_node = ox.nearest_nodes(G, start_latlon[1], start_latlon[0])
end_node = ox.nearest_nodes(G, end_latlon[1], end_latlon[0])

route_nodes = nx.shortest_path(G, start_node, end_node, weight='length')

# OSMnx の新しいルーティング関数を使って、正解ルートの緯度経度リストを取得
# 今回は可視化目的なので、resample_route_by_time は使いません。
# GDF からジオメトリを抽出し、緯度経度リストに変換します。
route_geom = ox.routing.route_to_gdf(G, route_nodes)
true_route_points = []
for geom in route_geom.geometry:
  if geom.geom_type == 'Point':
    true_route_points.append((geom.y, geom.x))
  elif geom.geom_type == 'LineString':
    for coord in list(geom.coords):
      true_route_points.append((coord[1], coord[0]))
  # MultiLineString の場合も考慮（複数のラインがある場合）
  elif geom.geom_type == 'MultiLineString':
    for line in geom.geoms:
      for coord in list(line.coords):
        true_route_points.append((coord[1], coord[0]))
# 重複を削除し、順番を保つ
unique_true_route_points = []
seen = set()
for p in true_route_points:
  if p not in seen:
    unique_true_route_points.append(p)
    seen.add(p)
print(f"正解ルートの計算が完了しました。")

# --- 地図を作成 ---
# データが存在する場合、その中心を地図の中心にする
if not df_detections.empty:
  map_center = [df_detections['latitude'].mean(), df_detections['longitude'].mean()]
else:
  map_center = [34.3422, 134.0463] # データがない場合は高松市役所

m = folium.Map(location=map_center, zoom_start=14)

print("地図上にデータをプロットしています...")

# --- 1. 正解ルート (緑色の太線) をプロット ---
if unique_true_route_points:
  folium.PolyLine(unique_true_route_points, color="darkgreen", weight=5, opacity=0.8, tooltip="正解ルート").add_to(m)


# --- 2. 検知データ（音の強弱に応じたグラデーション）をプロット ---
if not df_detections.empty:
  # 音の強さの最小値と最大値を取得 (0 除算を防ぐため min/max に補正)
  min_intensity = df_detections['sound_intensity'].min()
  max_intensity = df_detections['sound_intensity'].max()
  if max_intensity == min_intensity: # 全て同じ音量の場合
    norm = mcolors.NoNorm()
    cmap = cm.get_cmap('viridis', 1) # 単色
  else:
    norm = mcolors.Normalize(vmin=min_intensity, vmax=max_intensity)
    cmap = cm.get_cmap('viridis') # 青から黄色のグラデーション

  for _, row in df_detections.iterrows():
    # 音の強さに応じて色を決定
    rgba = cmap(norm(row['sound_intensity']))
    hex_color = mcolors.to_hex(rgba) # #RRGGBB 形式に変換

    folium.CircleMarker(
      location=[row['latitude'], row['longitude']],
      radius=2,
      color=hex_color,
      fill=True,
      fill_color=hex_color,
      fill_opacity=0.7,
      tooltip=f"音量: {row['sound_intensity']:.4f}"
    ).add_to(m)

# --- 3. 予測データ（赤い点と赤い線）をプロット ---
if not df_predictions.empty:
  # 予測位置を赤い「線」で結ぶ
  locations = df_predictions[['latitude', 'longitude']].values.tolist()
  folium.PolyLine(locations, color="red", weight=2.5, opacity=1, tooltip="予測ルート").add_to(m)

  # 予測位置を赤い「点」でプロットする
  for _, row in df_predictions.iterrows():
    folium.CircleMarker(
      location=[row['latitude'], row['longitude']],
      radius=3,
      color='red',
      fill=True,
      fill_color='red',
      fill_opacity=0.8,
      tooltip="予測位置"
    ).add_to(m)

# 凡例（カラーバー）を追加 (音量グラデーション用)
if not df_detections.empty and max_intensity > min_intensity:
  # カラーバーの HTML を生成
  colormap = cm.get_cmap('viridis')
  html_legend = """
  <div style="position: fixed;
        bottom: 50px; left: 50px; width: 150px; height: 120px;
        border:2px solid grey; z-index:9999; font-size:14px;
        background-color:white; opacity:0.9;">
```

```
    <div style="padding: 5px;"><b>音の強さ
</b></div>
    <div style="background: linear-gradient(to top,
  """

  # 0.0 から 1.0 までの音量に対応する色を 20 段階で生
成
  colors = [mcolors.to_hex(colormap(norm(i / 19 *
(max_intensity - min_intensity) + min_intensity)))
          for i in range(20)]
  colors_str = ", ".join([f"{c} {i*5}%" for i, c in
enumerate(colors)])

  html_legend += f"""
    {colors_str}
   );
    width:20px; height:80px; float:left; margin-
left:10px;"></div>
    <div style="float:right; margin-right:10px; margin-
top:5px;">
     <div>高 ({max_intensity:.2f})</div>
     <div style="height:60px;"></div>
     <div>低 ({min_intensity:.2f})</div>
    </div>
  </div>
  """

m.get_root().html.add_child(folium.Element(html_legend))


m.save(MAP_FILE_NAME)
print(f"\n 完了！ '{MAP_FILE_NAME}' が作成されまし
た。")
```

run_auto.sh(自動化スクリプト)

```
#!/bin/bash

# --- 設定エリア ---
SERVER_SCRIPT="main_map.py"          # WA サーバー
SIM_SCRIPT="simulator.py"        # WA シミュレータ
MODE_NAME="WA_MM"        # モード名
SIGMA_INFO="Physical=7.5m, GPS=15.0m"
# ▼▼▼ 保存先の設定 ▼▼▼
DIR_HTML="results_html"
DIR_CSV="results_csv"
SUMMARY_FILE="実験結果 WA.csv"

# 実験条件
SPEEDS=(40)
PHONES=(10)

# サーバーのポート番号
PORT=8003
START_ID=43
# ------------------

# フォルダ作成
if [ ! -d "$DIR_HTML" ]; then mkdir -p "$DIR_HTML"; fi
if [ ! -d "$DIR_CSV" ]; then mkdir -p "$DIR_CSV"; fi

echo
"==================================
========"
echo "  スマート自動実験 (待機機能付き) 開始"
echo "  モード: $MODE_NAME"
echo
"==================================
========"


CURRENT_ID=$START_ID

for SPEED in "${SPEEDS[@]}"; do
  for PHONE in "${PHONES[@]}"; do

    ID_STR=$(printf "%02d" $CURRENT_ID)

BASE_NAME="${ID_STR}_${MODE_NAME}_${SPEED}km_$
{PHONE}台"
    HTML_FILE="${DIR_HTML}/${BASE_NAME}.html"
    CSV_DETAIL="${DIR_CSV}/${BASE_NAME}.csv"

    echo ""
    echo "----------------------------------------"
    echo ">>> 実験 [ID: $ID_STR] : 速度=${SPEED}km/h,
台数=${PHONE}台"
    echo "----------------------------------------"

    # 1. DB 削除
    echo "[1/5] データベースリセット"
    if [ -f "locations.db" ]; then rm "locations.db"; fi

    # 2. サーバー起動
    echo "[2/5] サーバー起動開始 (地図 DL 待ち)..."
    # ログを server.log に書き出す（エラー確認用）
    uvicorn main_map:app --port $PORT --log-level info >
server_mm.log 2>&1 &
    SERVER_PID=$!

    # --- ★★★ 修正: Python を使った待機処理 ★★★ ---
    echo "    サーバーの応答を待っています..."

    MAX_RETRIES=60
    COUNT=0
    SERVER_READY=0

    while [ $COUNT -lt $MAX_RETRIES ]; do
      # Python ワンライナーで HTTP ステータスチェッ
ク
      # 成功(200)なら何も出力せず終了コード 0、失敗
なら終了コード 1
      python3 -c "import urllib.request; import sys;
sys.exit(0 if
urllib.request.urlopen('http://127.0.0.1:$PORT/').getcode(
) == 200 else 1)" > /dev/null 2>&1

      # 直前のコマンド($?)が成功(0)ならループを抜け
る
      if [ $? -eq 0 ]; then
        SERVER_READY=1
        break
      fi
```

```
        sleep 2
        COUNT=$((COUNT + 1))
        echo -n "."
    done

    echo ""

    if [ $SERVER_READY -eq 1 ]; then
        echo "    -> サーバー準備完了！ (PID: $SERVER_PID)"
    else
        echo "    ❌ エラー: サーバーが起動しませんでした"
        echo "    server.log を確認してください"
        kill $SERVER_PID
        exit 1
    fi
    # --- ▲▲▲ 修正ここまで ▲▲▲ ---

    # 3. シミュレーション実行
    echo "[3/5] シミュレーション実行中..."
    python "$SIM_SCRIPT" --speed "$SPEED" --phones "$PHONE"
    echo "    -> 完了"

    # 4. サーバー停止
    kill $SERVER_PID
    sleep 2

    # 5. Visualize
    echo "[4/5] 地図作成..."
    python visualize.py --output "$HTML_FILE"
    echo "    -> $HTML_FILE"

    # 6. Evaluate
    echo "[5/5] 評価・集計..."
    python evaluate.py \
        --speed "$SPEED" \
        --phones "$PHONE" \
        --mode "$MODE_NAME" \
        --output_detail "$CSV_DETAIL" \
        --append_summary "$SUMMARY_FILE" \
        --exp_id "$ID_STR"\
        --sigma_info "$SIGMA_INFO"

    echo "    -> 完了 (ID: $ID_STR)"

    CURRENT_ID=$((CURRENT_ID + 1))

  done
done

echo ""
echo "================================================"
echo "  全実験終了！"
echo "================================================"
```

クライアントアプリケーション

```
// MainActivity.kt (音量表示機能付き)

package com.example.emergencyvehiclenotifier // ★ あなたのパッケージ名に合わせてください

// (必要な import 文は省略。Android Studio が自動で追加してくれます)
import android.Manifest
import android.annotation.SuppressLint // SuppressLint を使うために必要
import android.content.Context
import android.content.pm.PackageManager
import android.graphics.Color
import android.location.Location
import android.media.*
import android.os.*
import android.util.Log
import android.widget.TextView
import android.widget.Toast
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AppCompatActivity
import androidx.constraintlayout.widget.ConstraintLayout
import androidx.core.content.ContextCompat
import com.google.android.gms.location.*
import com.google.android.gms.maps.CameraUpdateFactory
import com.google.android.gms.maps.GoogleMap
import com.google.android.gms.maps.MapView
import com.google.android.gms.maps.OnMapReadyCallback
import com.google.android.gms.maps.model.BitmapDescriptorFactory
import com.google.android.gms.maps.model.LatLng
import com.google.android.gms.maps.model.Marker
import com.google.android.gms.maps.model.MarkerOptions
import kotlinx.coroutines.*
import org.jtransforms.fft.DoubleFFT_1D
import retrofit2.Response
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import retrofit2.http.Body
import retrofit2.http.GET
import retrofit2.http.POST
import java.util.LinkedList
import kotlin.math.*

// --- サーバー通信用のデータクラス ---
data class LocationData(
    val latitude: Double,
    val longitude: Double,
    val sound_intensity: Double
)

data class PredictionResponse(
    val status: String,
    val latitude: Double?,
    val longitude: Double?
)

// --- Retrofit API インターフェース ---
interface ApiService {
    @POST("location")
```

```
    suspend fun sendLocation(@Body location:
LocationData): Response<Unit>

    @GET("latest_prediction")
    suspend fun getLatestPrediction():
Response<PredictionResponse>
//    @GET("predict")
//    suspend fun triggerAndGetPrediction():
Response<PredictionResponse> // 関数名を変更
}

class MainActivity : AppCompatActivity(),
OnMapReadyCallback {

    // --- 定数 ---
    private val TAG = "MainActivity"
    private val POLLING_INTERVAL_MS = 5000L
    private val ALERT_DISTANCE_METERS = 500.0
    private val MAX_AMPLITUDE = 15000.0
    private val DEFAULT_LOCATION = LatLng(34.34,
134.04)
    private val DEFAULT_ZOOM = 14f

    // --- 音声処理関連 ---
    private val sampleRate = 44100
    private val bufferSizeInBytes =
AudioRecord.getMinBufferSize(sampleRate,
AudioFormat.CHANNEL_IN_MONO,
AudioFormat.ENCODING_PCM_16BIT)
    private val bufferSizeInShorts = if (bufferSizeInBytes >
0) bufferSizeInBytes / 2 else 1024
    private var audioRecord: AudioRecord? = null
    private var isRecording = false
    private var audioJob: Job? = null
    private val sirenFreqMin = 700.0
    private val sirenFreqMax = 1500.0
    private val freqHistory = LinkedList<Double>()
    private val historySize = 5

    // --- UI 要素 ---
    private lateinit var statusTextView: TextView
    private lateinit var distanceTextView: TextView
    private lateinit var mainLayout: ConstraintLayout
    private lateinit var mapView: MapView
    private var googleMap: GoogleMap? = null
    private var predictionMarker: Marker? = null
    // ▼▼▼ 追加 ▼▼▼
    private lateinit var currentAmplitudeTextView: TextView
    private lateinit var sentIntensityTextView: TextView
    // ▲▲▲ 追加 ▲▲▲

    // --- 位置情報 & 通信 ---
    private lateinit var fusedLocationClient:
FusedLocationProviderClient
    private var currentLocation: Location? = null
    private val apiService: ApiService by lazy {
        Retrofit.Builder()
            //
★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★
            //★ ここを、あなたの PC の IP アドレスに書き換
えてください ★
            //
★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★
            .baseUrl("http://192.168.10.5:8000/")
            .addConverterFactory(GsonConverterFactory.create(
))
            .build()
            .create(ApiService::class.java)
    }

    // --- ポーリング（定期実行）関連 ---
    private val handler = Handler(Looper.getMainLooper())
    private var pollingJob: Job? = null

    // --- 権限リクエストランチャー ---
    private val permissionLauncher =
registerForActivityResult(
        ActivityResultContracts.RequestMultiplePermissions()
    ) { permissions ->
        val granted = permissions.entries.all { it.value }
        if (granted) {
            Log.d(TAG, "権限が付与されました")
            initializeComponents()
        } else {
            Toast.makeText(this, "マイクと位置情報の権限が必
要です", Toast.LENGTH_LONG).show()
            Log.e(TAG, "権限が拒否されました")
            statusTextView.text = "権限が必要です"
        }
    }

    // --- アクティビティライフサイクル ---
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        Log.d(TAG, "onCreate")

        // UI 要素を取得
        statusTextView = findViewById(R.id.statusTextView)
        distanceTextView =
findViewById(R.id.distanceTextView)
        mainLayout = findViewById(R.id.mainLayout)
        mapView = findViewById(R.id.mapView)
        // ▼▼▼ 追加 ▼▼▼
        currentAmplitudeTextView =
findViewById(R.id.currentAmplitudeTextView)
        sentIntensityTextView =
findViewById(R.id.sentIntensityTextView)
        // ▲▲▲ 追加 ▲▲▲


        mapView.onCreate(savedInstanceState)
        mapView.getMapAsync(this)

        fusedLocationClient =
LocationServices.getFusedLocationProviderClient(this)

        checkPermissionsAndStart()
    }

    override fun onMapReady(map: GoogleMap) {
        googleMap = map
        Log.d(TAG, "地図の準備完了")

googleMap?.moveCamera(CameraUpdateFactory.newLatLn
gZoom(DEFAULT_LOCATION, DEFAULT_ZOOM))
        enableMyLocation()
```

```
    }

    // --- 初期化と権限チェック ---
    private fun checkPermissionsAndStart() {
        Log.d(TAG, "権限を確認中...")
        val permissionsNeeded = arrayOf(
            Manifest.permission.RECORD_AUDIO,
            Manifest.permission.ACCESS_FINE_LOCATION,
            Manifest.permission.ACCESS_COARSE_LOCATION
        ).filter {
            ContextCompat.checkSelfPermission(this, it) !=
PackageManager.PERMISSION_GRANTED
        }

        if (permissionsNeeded.isNotEmpty()) {
            Log.d(TAG, "権限をリクエスト:
$permissionsNeeded")

permissionLauncher.launch(permissionsNeeded.toTypedA
rray())
        } else {
            Log.d(TAG, "必要な権限はすべて付与されています
")
            initializeComponents()
        }
    }

    private fun initializeComponents() {
        Log.d(TAG, "コンポーネントを初期化中...")
        statusTextView.text = "待機中..."
        startLocationUpdates()
        startAudioProcessing()
        startPolling()
        enableMyLocation()
    }

    @SuppressLint("MissingPermission") // 権限チェック
は呼び出し元で行う
    private fun enableMyLocation() {
        if (hasLocationPermission()) {
            try {
                googleMap?.isMyLocationEnabled = true
                Log.d(TAG, "地図の現在地レイヤー有効化")
            } catch (e: SecurityException) {
                Log.e(TAG, "地図の現在地表示で権限エラー", e)
            }
        } else {
            Log.w(TAG, "現在地レイヤー有効化不可：権限なし
")
        }
    }

    // --- 位置情報の継続取得 ---
    @SuppressLint("MissingPermission")
    private fun startLocationUpdates() {
        if (!hasLocationPermission()) return

        val locationRequest =
LocationRequest.Builder(Priority.PRIORITY_HIGH_ACCUR
ACY, 5000L)
            .setMinUpdateIntervalMillis(2000L)
            .build()

        val locationCallback = object : LocationCallback() {
            override fun onLocationResult(locationResult:
LocationResult) {
                locationResult.lastLocation?.let {
                    currentLocation = it
                    Log.v(TAG, "現在地更新: ${it.latitude},
${it.longitude}")
                }
            }
        }
        try {

fusedLocationClient.requestLocationUpdates(locationRequ
est, locationCallback, Looper.getMainLooper())
            Log.d(TAG, "位置情報の更新を開始しました")
        } catch(e: SecurityException) {
            Log.e(TAG, "位置情報更新リクエスト失敗", e)
        } catch (e: Exception){
            Log.e(TAG, "位置情報開始時の予期せぬエラー", e)
        }
    }

    // --- 音声処理 & サイレン検知 ---
    @SuppressLint("MissingPermission")
    private fun startAudioProcessing() {
        if (!hasRecordAudioPermission() || isRecording)
return

        try {
            audioRecord =
AudioRecord(MediaRecorder.AudioSource.MIC,
sampleRate, AudioFormat.CHANNEL_IN_MONO,
AudioFormat.ENCODING_PCM_16BIT, bufferSizeInBytes)
            if (audioRecord?.state !=
AudioRecord.STATE_INITIALIZED) {
                Log.e(TAG, "AudioRecord 初期化失敗") ; return
            }
            audioRecord?.startRecording()
            isRecording = true
            Log.d(TAG, "音声録音開始")

            audioJob = CoroutineScope(Dispatchers.IO).launch {

Process.setThreadPriority(Process.THREAD_PRIORITY_UR
GENT_AUDIO)
                val audioBuffer = ShortArray(bufferSizeInShorts)
                val fft =
DoubleFFT_1D(bufferSizeInShorts.toLong())
                val fftBuffer = DoubleArray(bufferSizeInShorts *
2)

                while (isActive && isRecording) {
                    val readResult = audioRecord?.read(audioBuffer,
0, bufferSizeInShorts) ?:
AudioRecord.ERROR_INVALID_OPERATION
                    if (readResult > 0) {
                        val rmsAmplitude =
calculateRMS(audioBuffer, readResult)
                        val dominantFreq = performFFT(audioBuffer,
readResult, fft, fftBuffer)

                        // ▼▼▼ 追加 ▼▼▼
                        // UI スレッドで現在の音量を表示
                        withContext(Dispatchers.Main) {
                            currentAmplitudeTextView.text =
"%.1f".format(rmsAmplitude)
                        }
```

```
                // ▲▲▲ 追加 ▲▲▲

                if (isSirenDetected(dominantFreq)) {
                    Log.i(TAG, "サイレン検知！ Freq: %.1f Hz,
RMS: %.1f".format(dominantFreq, rmsAmplitude))
                    withContext(Dispatchers.Main)
{ statusTextView.text = "サイレン検知！ データ送信
中..." }
                    sendSirenDataToServer(rmsAmplitude)
                    delay(100)
                    freqHistory.clear()
                    withContext(Dispatchers.Main)
{ statusTextView.text = "待機中..." }
                }
            } else if (readResult < 0) {
                Log.w(TAG, "AudioRecord 読み込みエラー:
$readResult")
                isRecording = false
            }
        }
        Log.d(TAG, "音声処理コルーチン終了")
        // コルーチン終了時にリソース解放
        withContext(Dispatchers.Main) { // UI スレッド
でリソース解放は避けるべきだが、サンプルとして
            audioRecord?.stop()
            audioRecord?.release()
            audioRecord = null
            isRecording = false
            Log.d(TAG, "AudioRecord リソース解放")
        }
    }
    } catch (e: SecurityException) {
        Log.e(TAG, "AudioRecord 権限エラー", e)
    } catch (e: Exception) {
        Log.e(TAG, "音声処理開始失敗", e)
    }
  }

  // RMS 計算
  private fun calculateRMS(buffer: ShortArray, readSize:
Int): Double {
    if (readSize <= 0) return 0.0
    var sumSquare = 0.0
    for (i in 0 until readSize) {
        sumSquare += buffer[i] * buffer[i]
    }
    return sqrt(sumSquare / readSize)
  }

  // FFT 実行と支配周波数計算
  private fun performFFT(audioBuffer: ShortArray,
readSize: Int, fft: DoubleFFT_1D, fftBuffer: DoubleArray):
Double {
    for(i in 0 until readSize) fftBuffer[i] =
audioBuffer[i].toDouble()
    for(i in readSize until bufferSizeInShorts) fftBuffer[i] =
0.0
    fft.realForward(fftBuffer)

    var maxMagnitude = -1.0
    var dominantFreq = 0.0
    for (i in 0 until bufferSizeInShorts / 2) {
        val real = if (i == 0) fftBuffer[0] else fftBuffer[2 * i]
        val imag = if (i == 0) 0.0 else fftBuffer[2 * i + 1]
        val magnitude = sqrt(real * real + imag * imag)
        if (magnitude > maxMagnitude) {
            maxMagnitude = magnitude
            dominantFreq = i * sampleRate.toDouble() /
bufferSizeInShorts
        }
    }
    return dominantFreq
  }


  // サイレン判定ロジック
  private fun isSirenDetected(dominantFreq: Double):
Boolean {
    val inRange = dominantFreq in
sirenFreqMin..sirenFreqMax
    if (!inRange) return false

    freqHistory.add(dominantFreq)
    if (freqHistory.size > historySize)
freqHistory.removeFirst()

    val distinctFreqCount = freqHistory.distinctBy
{ round(it / 50.0) * 50.0 }.size
    return distinctFreqCount > 1
  }

  // --- サーバー通信 ---
  private fun sendSirenDataToServer(amplitude: Double) {
    val loc = currentLocation ?: run {
        Log.w(TAG, "データ送信不可：現在地不明")
        runOnUiThread { Toast.makeText(this, "現在地が取
得できません", Toast.LENGTH_SHORT).show() }
        return
    }

    val normalizedIntensity = (amplitude /
MAX_AMPLITUDE).coerceIn(0.0, 1.0)
    Log.d(TAG, "生振幅: %.1f, 正規化
後: %.3f".format(amplitude, normalizedIntensity))

    // ▼▼▼ 追加 ▼▼▼
    // UI スレッドで送信した音量を表示
    runOnUiThread {
        sentIntensityTextView.text =
"%.3f".format(normalizedIntensity)
    }
    // ▲▲▲ 追加 ▲▲▲

    val data = LocationData(
        latitude = loc.latitude,
        longitude = loc.longitude,
        sound_intensity = normalizedIntensity
    )

    CoroutineScope(Dispatchers.IO).launch {
        try {
            val response = apiService.sendLocation(data)
            if (response.isSuccessful) {
                Log.i(TAG, "サーバーへのデータ送信成功")
            } else {
                Log.w(TAG, "サーバーエラー:
${response.code()}")
```

```
            }
        } catch (e: Exception) {
            Log.e(TAG, "サーバーへの送信失敗", e)
        }
    }
  }

  // --- 定期的な予測位置の取得（ポーリング） ---
  private fun startPolling() {
    if (pollingJob?.isActive == true) return
    Log.d(TAG, "ポーリング開始")
    pollingJob =
CoroutineScope(Dispatchers.Default).launch {
        while(isActive) {
            fetchLatestPrediction()
            delay(POLLING_INTERVAL_MS)
        }
        Log.d(TAG, "ポーリング終了")
    }
  }

  private suspend fun fetchLatestPrediction() {
    try {
        val response = apiService.getLatestPrediction()
        //val response =
apiService.triggerAndGetPrediction()

        if (response.isSuccessful && response.body() !=
null) {
            val prediction = response.body()!!
//          if (response.isSuccessful && response.body() !=
null) {
//              val prediction = response.body()!!


            if (prediction.status == "success" &&
prediction.latitude != null && prediction.longitude != null)
{
                val predictionLatLng =
LatLng(prediction.latitude, prediction.longitude)
                Log.d(TAG, "予測位置受信:
${prediction.latitude}, ${prediction.longitude}")
                withContext(Dispatchers.Main) {
                    updatePredictionMarker(predictionLatLng)
                    checkDistanceAndAlert(predictionLatLng)
                }
            } else if (prediction.status == "nodata") {
                Log.d(TAG, "予測データなし")
                withContext(Dispatchers.Main) { clearAlert();
distanceTextView.text = "N/A" }
            } else {
                Log.w(TAG, "予測取得サーバーエラー:
${prediction.status}")
                withContext(Dispatchers.Main) { clearAlert();
distanceTextView.text = "サーバーエラー" }
            }
        } else {
            Log.w(TAG, "予測取得失敗: ${response.code()}")
            withContext(Dispatchers.Main) { clearAlert();
distanceTextView.text = "通信失敗" }
        }
    } catch (e: Exception) {
        Log.e(TAG, "予測取得通信エラー", e)
        withContext(Dispatchers.Main) { clearAlert();
distanceTextView.text = "通信エラー" }
    }
  }

  // --- 地図マーカーと接近警告 ---
  private fun updatePredictionMarker(latLng: LatLng) {
    googleMap?.let { map ->
        if (predictionMarker == null) {
            predictionMarker = map.addMarker(
                MarkerOptions()
                    .position(latLng)
                    .title("予測位置")
                    .anchor(0.5f, 0.5f)
            )
        } else {
            predictionMarker?.position = latLng
        }
    }
  }

  private fun checkDistanceAndAlert(predictionLatLng:
LatLng) {
    val currentLoc = currentLocation ?: run {
        distanceTextView.text = "N/A (現在地不明)"
        clearAlert()
        return
    }

    val results = FloatArray(1)
    Location.distanceBetween(
        currentLoc.latitude, currentLoc.longitude,
        predictionLatLng.latitude,
predictionLatLng.longitude,
        results
    )
    val distance = results[0]
    distanceTextView.text = "%.1f m".format(distance)
    Log.d(TAG,"予測位置までの距離: $distance m")

    if (distance <= ALERT_DISTANCE_METERS) {
        showAlert()
    } else {
        clearAlert()
    }
  }

  private fun showAlert() {
    if (!statusTextView.text.contains("接近中")) {
        statusTextView.text = "!! 緊急車両 接近中 !!"
        mainLayout.setBackgroundColor(Color.RED)
        vibratePhone()
        Log.w(TAG, "警告表示!")
    }
  }

  private fun clearAlert() {
    if (statusTextView.text.contains("接近中")) {
        if (statusTextView.text.contains("サイレン検知")) {
            // Keep siren detected status
        } else {
            statusTextView.text = "待機中..."
        }
        mainLayout.setBackgroundColor(Color.WHITE)
```

```
            Log.d(TAG, "警告解除")
        }
    }

    private fun vibratePhone() {
        val vibrator =
getSystemService(Context.VIBRATOR_SERVICE) as
Vibrator
        if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.O) {
            vibrator.vibrate(VibrationEffect.createOneShot(500,
VibrationEffect.DEFAULT_AMPLITUDE))
        } else {
            vibrator.vibrate(500)
        }
    }

    // --- 権限チェックヘルパー ---
    private fun hasRecordAudioPermission() =
        ContextCompat.checkSelfPermission(this,
Manifest.permission.RECORD_AUDIO) ==
PackageManager.PERMISSION_GRANTED

    private fun hasLocationPermission() =
        ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED ||
            ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) ==
PackageManager.PERMISSION_GRANTED


    // --- アクティビティライフサイクル管理（MapView
に必須） ---
    // (変更なし。onStart, onResume, onPause, onStop,
onDestroy, onLowMemory, onSaveInstanceState を含む)
    override fun onStart() { super.onStart();
mapView.onStart() }
    override fun onResume() {
        super.onResume()
        mapView.onResume()
        if (hasRecordAudioPermission() &&
hasLocationPermission()) {
            startPolling()
            if (!isRecording) startAudioProcessing()
        }
        Log.d(TAG, "onResume")
    }
    override fun onPause() {
        super.onPause()
        mapView.onPause()
        pollingJob?.cancel() // ポーリング停止
        Log.d(TAG, "onPause, ポーリング停止")
    }
    override fun onStop() { super.onStop();
mapView.onStop() }
    override fun onDestroy() {
        super.onDestroy()
        mapView.onDestroy()
        isRecording = false
        audioJob?.cancel() // 音声処理コルーチン停止
        pollingJob?.cancel() // ポーリングコルーチン停止
        audioRecord?.release() // リソース解放
        Log.d(TAG, "onDestroy")
    }
    override fun onLowMemory() { super.onLowMemory();
mapView.onLowMemory() }
    override fun onSaveInstanceState(outState: Bundle) {
        super.onSaveInstanceState(outState)
        mapView.onSaveInstanceState(outState)
    }
} // MainActivity クラスの終わり
```