平成 30 年度　特別研究報告書

広域無線通信技術による
山岳遭難事故防止に関する提案

龍谷大学　理工学部　情報メディア学科

T150510　辻 広人

指導教員　三好 力　教授

内容梗概

近年、登山スタイルの変化や自然志向の高まりから携帯電話やインターネットの普及で下火になっていたアウトドアブームが再燃している。一方で、高齢者層やアウトドアの初心者層を中心に山岳地帯の遭難事故が年々増加している。昨今ではスマートフォンなどの通信端末の普及が進んでいるが、遭難事故のうち 25 パーセントほどが遭難現場から救助要請がおこなわれていないのが現況である。多くの山岳では発達した情報技術が行き届いていないため電波が確保されていないことやバッテリー切れのリスクがあるなど十分な注意を払うことが登山では必要不可欠である。

本論文では、多くの山岳では発達した通信技術が行き届いていない点に着目し、省電力で長距離通信を可能にする LPWA（Low Power, Wide Area）技術を用いて山岳地帯全体にネットワーク環境の構築をおこない、情報技術分野から山岳遭難事故の防止を支援するシステムを提案する。これにより、公衆電話などの電話回線やネットワーク回線が確実に繋がっている機器をゲートウェイとして介すことで、携帯電話や無線などの通信手段を使用できない状況のなか山岳地帯で遭難した場合でも救助要請をおこなうことが期待される。

目次

第1章 はじめに

1.1 研究背景

近年、環境保全への関心の高まりや自然志向のライフスタイルなどの要因から携帯電話やインターネットの普及で下火になっていたアウトドアブームが再燃している。その背景には登山スタイルやキャンプスタイルの変化があり、その後押しとなっているのが SNS の流行である。Facebook や Instagram に投稿する写真を撮影するために、喧騒から離れた非日常の登山やキャンプへの人気が女性を中心に高まった。矢野経済研究所の調査によると、アウトドア用品や関連施設などの市場を合計した 2017 年の国内アウトドア市場は前年比 103.2%の 4398.3 億円となった。図 1 にその内訳を示す。[1] キャンプ、ハイキング、野外フェスなどが含まれるライトアウトドア分野をはじめとした各分野の市場規模が拡大している。それぞれ堅調に推移しており、今後も幅広い消費者層の取り込みでさらに市場が好調になると推測される。

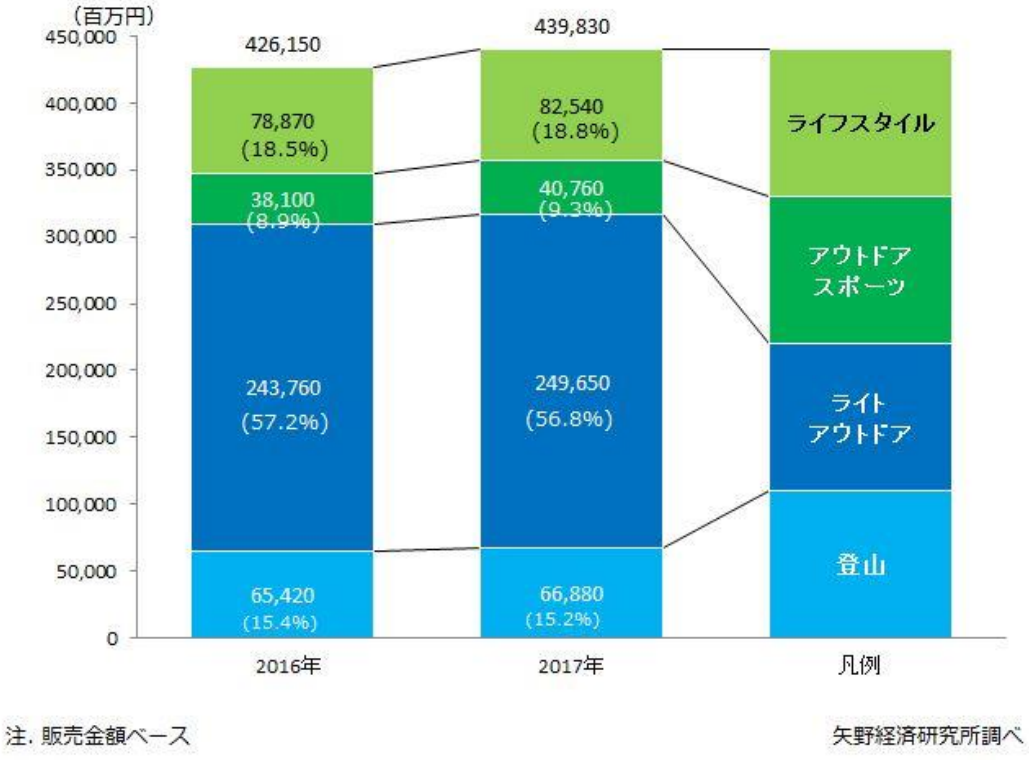

図 1 国内アウトドア市場の推移と内訳

一方で警察庁の統計[2]によると、山岳地帯の遭難事故が増加傾向にあり、2017 年から過去 10 年間で 952 件（+58.3%）増加していることが表 1 から読み取れる。表 2 によると特に高齢者層に山岳遭難者が多く、60 歳以上の世代が全体の 50%ほどを占めていることがわかる。しかし、20 代以下の山岳遭難者も増加傾向にあり、今後さらにその比率が高くなっていくことが予測される。これらの原因として、高齢者層の体力の低下や若年層の登山への準備不足などが挙げられる。表 3 から、昨今の携帯電話やスマートフォンの普及などによって 2017 年に発生した遭難事故のうち 77.5%が遭難現場から救助要請をおこなっていることがわかるが、多くの山岳では発達した情報技術が行き届いていないため電波が確保されていないことやバッテリー切れのリスクがあるなど十分な注意が必要である。

本研究では、省電力で長距離通信を可能にする LPWA（Low Power, Wide Area）技術を用いて山岳地帯全体にネットワーク環境を構築し、情報技術分野から山岳遭難事故の発生を防止することを目標とする。これにより、公衆電話などの電話回線やネットワーク回線が確実に繋がっている機器をゲートウェイとして介すことで、携帯電話や無線などの通信手段を使用できない状況のなか山岳地帯で遭難した場合でも救助要請をおこなうことが期待される。

表 1 山岳遭難事故発生件数の推移

	平成20年	平成21年	平成22年	平成23年	平成24年	平成25年	平成26年	平成27年	平成28年	平成29年	構成比
発生件数（件）	1,631	1,676	1,942	1,830	1,988	2,172	2,293	2,508	2,495	2,583	
遭難者数（人）	1,933	2,085	2,396	2,204	2,465	2,713	2,794	3,043	2,929	3,111	100.0%
死者・行方不明者	281	317	294	275	284	320	311	335	319	354	11.4%
死者	253	269	262	244	249	278	272	298	278	315	10.1%
行方不明者	28	48	32	31	35	42	39	37	41	39	1.3%
負傷者	698	670	832	819	927	1,003	1,041	1,151	1,133	1,208	38.8%
無事救出者	954	1,098	1,270	1,110	1,254	1,390	1,442	1,557	1,477	1,549	49.8%

表２ 年齢層別山岳遭難者の推移

	平成25年	平成26年	平成27年	平成28年	平成29年	
	人数	人数	人数	人数	人数	構成比
20 歳 未 満	230	153	201	174	189	6.1%
20 ～ 29	236	222	228	194	261	8.4%
30 ～ 39	251	281	277	291	240	7.7%
40 ～ 49	332	333	372	366	378	12.2%
50 ～ 59	406	402	397	421	455	14.6%
60 ～ 69	686	744	791	746	741	23.8%
70 ～ 79	466	537	609	565	669	21.5%
80 ～ 89	97	114	151	161	165	5.3%
90 歳 以 上	9	6	14	10	13	0.4%
不 明		2	3	1		0.0%
合 計	2,713	2,794	3,043	2,929	3,111	100.0%

表３ 通信手段の使用状況の推移

		平成25年	平成26年	平成27年	平成28年	平成29年	
		件数	件数	件数	件数	件数	構成比
発 生 件 数		2,172	2,293	2,508	2,495	2,583	
使 用 あ り		1,548	1,728	1,922	1,907	2,003	77.5%
	携 帯 電 話	1,527	1,707	1,920	1,905	1,991	77.1%
	無 線	21	21	2	2	12	0.5%
使 用 な し		624	565	586	588	580	22.5%

注1::通話エリア圏外、バッテリー切れ等は「使用なし」に含む。

注2:携帯電話・無線機併用は、無線機に計上。

第2章 既存技術と問題点

2.1 LPWA について

2.1.1 LPWA とは

LPWA は、これまでの通信技術が目指していた高速化や大容量化ではなく、少ない電力(Low Power)で広い領域(Wide Area)の通信を実現する IoT 社会に特化した無線通信システムであり、一般的な電池による数年から数十年にわたっての運用や数 km から数十 km もの広域通信が可能になる。既存の無線通信規格である Bluetooth や無線 LAN、LTE などはそもそも IoT/M2M での利用を想定して設計されていないため、LWPA が今後の IoT 市場を牽引するシステムになることが期待されている。[3] 無線通信規格における LPWA の立ち位置を図 2 に示す。

接続可能台数をシンプルな通信処理で増やすことができ、これらは低ビットレート通信により実現している。システムが状況を認識するために必要な情報をサーバーに送るだけなので IoT/M2M では高速通信を重要視していないため、主に扱うデータは軽量で通信コストも非常に低い。総務省[4]によると、これまで実現が不可能だった領域への導入することで用途に応じた無線帯域のインフラが進むと予測されており、2021 年には 3 億 8,000 万台まで拡大、売上金額 10 億ドル規模になるとみられている。図 3 は 2016 年までの推移と今後の予測を示した図である。

LPWA の通信規格は明確に決まっておらず、主に既存の LTE 通信規格を基にしたセルラー系規格と独自に開発された非セルラー系 LPWA があり、用途に応じて LPWA 規格を選択することができる。セルラー系 LPWA は全国規模のネットワーク展開が容易で通信も安定しており通信速度も高速だが、規格が複雑であることから通信料金や消費電力が大きいことが欠点である。対して、非セルラー系 LPWA はシンプルな規格のものが多く、デバイス価格や通信料金が従来の IoT 通信プランの月額数十円～数百円と比較して数円～数十円と非常に安価である。また、免許が不要な周波数を基本的に使用しているため事業参入しやすく、主要 LPWA のほとんどが非セルラー系規格である。

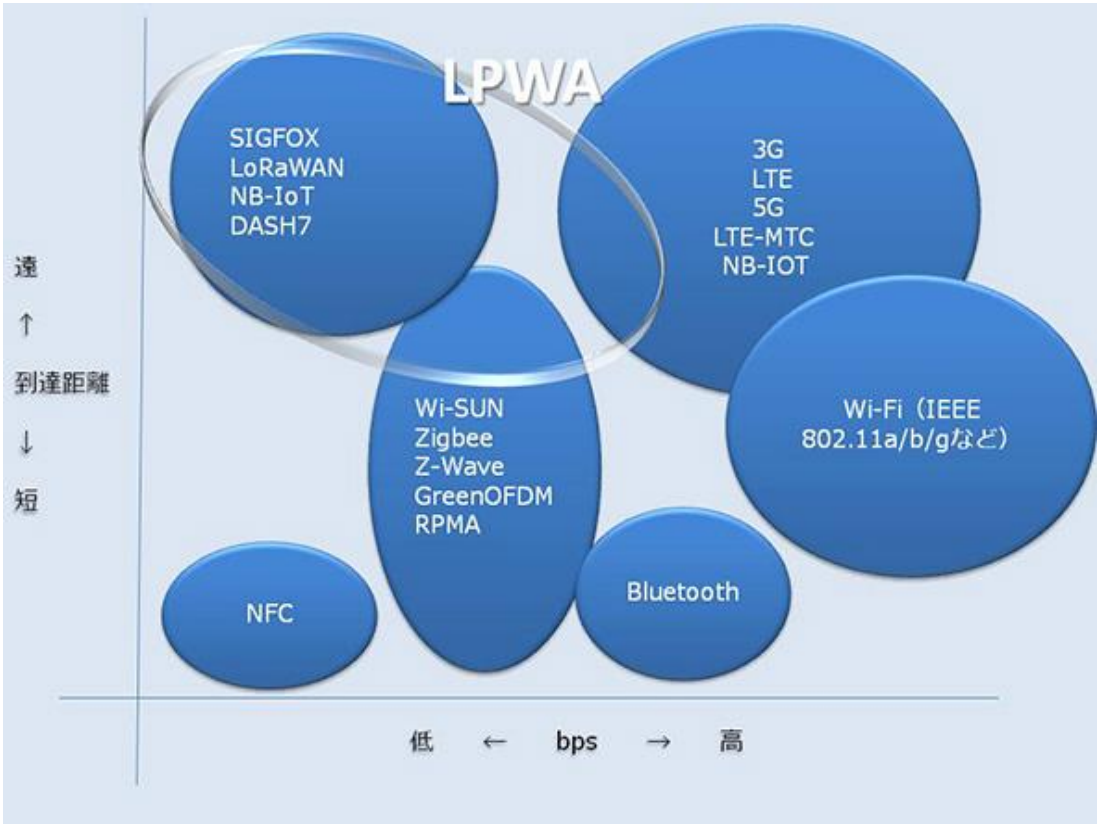

図 2　無線通信規格における LPWA の立ち位置

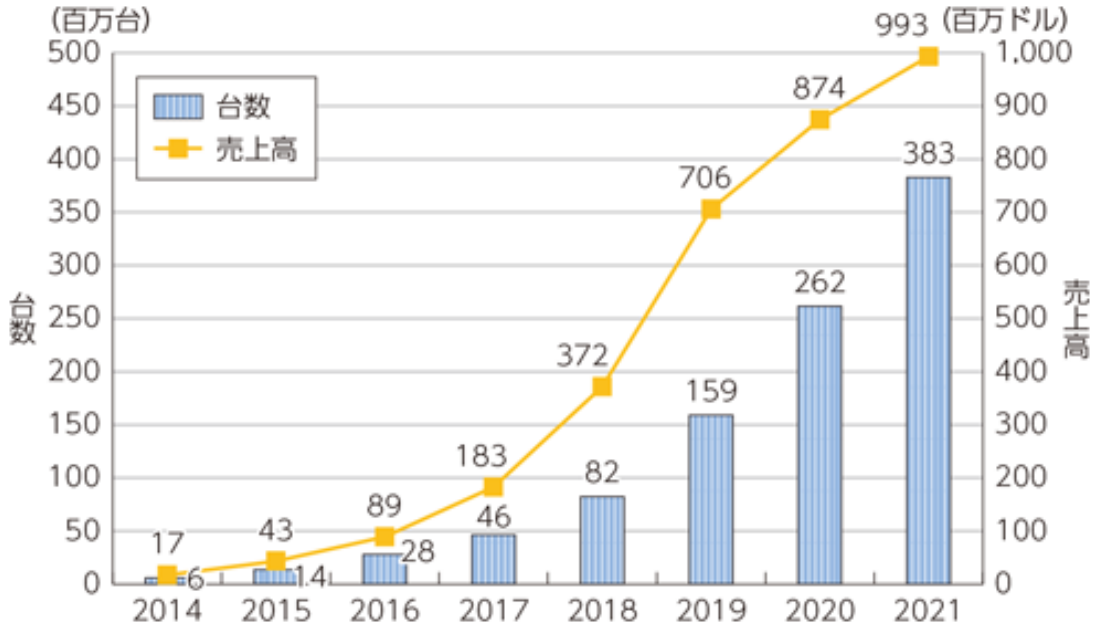

図 3　LPWA の台数及び接続売上高の推移と予測

2.1.2　LPWA の種類

・LoRa/LoRaWAN [5][6]

LoRa とは物理層に直接拡散方式のスペクトラム拡散を用いた変調方式のことで、信号を拡散・復元してノイズの影響を低減する。LoRaWAN は LoRa 変調を用いたデータ送受信の仕様で、図 4 のように LoRa デバイスはどの LoRa ゲートウェイに対しても接続をおこない、ネットワークサーバーにデータを伝達するオープンな通信規約である。特定都市の都市インフラや民間の土地、施設用インフラなどの限定されたエリア内での IoT ネットワーク構築に向いている。

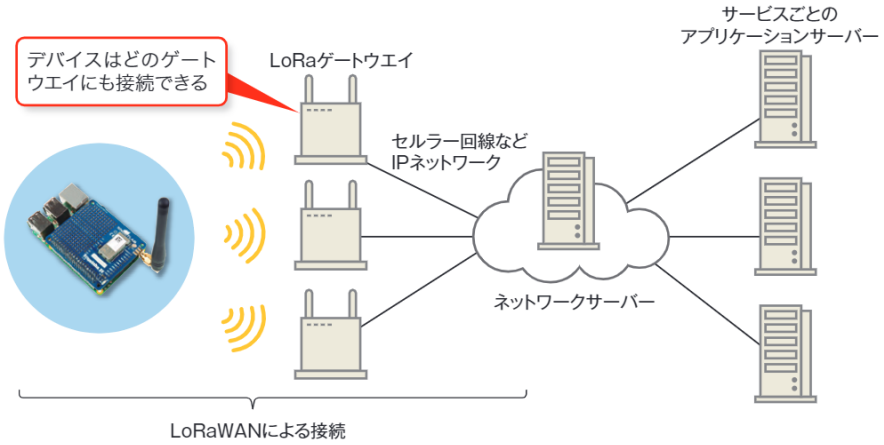

図 4　LoRaWAN のイメージ図

・SIGFOX [5][6]

一つのデバイスからの電波を複数の基地局で受信することで、ある基地局に届いた信号の一部がノイズで隠れてしまっても、ほかの基地局に届くデータから復元する。また、超狭帯域の信号を使用しているため、ノイズの多い帯域でも信号の識別が容易である。他の LPWA 規格と比較しても消費電力や価格を抑え、最大数百 km の広いエリアをカバーできる。クラウドサービスも提供しておりデータの解析が可能であるなど、これらの特徴から主に IoT 広域ネットワークの構築や全国で利用される製品への実装に向いている。

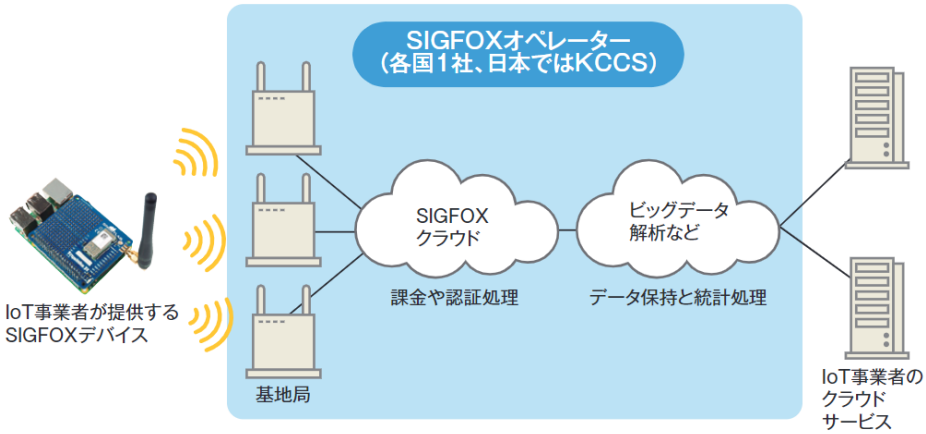

図 5　SIGFOX のイメージ図

・Wi-SUN

厳密には LPWA には含まれないが、スマートメーターから無線通信をつかって効率的に検針データを収集する無線通信システムであるため同様の目的を果たしている。LPWA と同じく低消費電力かつ低速通信だが、1 台あたりの通信可能距離は 500m 程度のためデバイス同士が通信を中継しあうマルチホップルーティングを用いて長距離の通信に対応している。2.4GHz 帯やサブ GHz 帯に加え複数の変調方式を利用しているため、障害物や干渉に強い。

2.1.3 問題点

デバイスに電源供給が常に必要になるものが多かったことや Wi-Fi や Bluetooth など最大通信可能距離が 1km 未満のものが多かった従来の IoT/M2M の問題を省電力で長距離広域通信ができる LPWA で解決した反面、通信速度は最大 1Mbps 程度と携帯電話システムなどと比較して低速となっており、今後大きいデータを扱わなければならない場面で問題が生じる。

また、LPWA は実際の通信環境によって変動し、障害物などのない見通しの良い土地であれば数十 km の通信範囲が確保できるが、高層ビルが立ち並ぶ都市部や木が生い茂る森林部、また移動中では 1km 程度にとどまる場合もある。通信回数や通信量による消費電力量も変化するので、通信頻度などの通信環境や利用環境から理論上の値が出ない場合が多い。

2.2 TREK TRACK

2.2.1 特徴

TREK TRACK は、登山者の位置情報の可視化と集中管理をおこない山岳事故の減少を実現するサービスである。[7]

図 4 のように、事前にレンタルしたデバイスを携行するだけで GPS 情報がサーバーに送信されるため位置情報をリアルタイムに把握でき、デバイスから救助要請が出された際に速やかに対応することができる。ゲートウェイは主に山小屋に設置され、電源が確保できない場所でもソーラーバッテリーにより対応している。

また、収集した GPS 情報から行動履歴データが蓄積されるため、滞留しやすい登山道の整備やマーケティング調査の実施が可能である。スキー場での活用もおこなわれている。

2.2.2 問題点

デバイスを事前レンタルするという手間がかかり、さらに自分が行くエリアがシステムの対象エリアであるか調べる必要が生じる。過去発生している山岳事故の原因の多くは準備不足によるものなので、そうした登山者層の意識の改善なしには山岳事故の減少に繋がらないと推測される。

また、現在このシステムを利用できる山岳は少なく、これから登山者や利用者が多い山岳からサービス対応されることが予想されるが、登山者が少なく発見が困難な山岳ほど必要である。

情報が可視化されることで位置が特定できる反面、ただ休憩しているだけでも救助要請が出せない状況と勘違いしてしまう可能性が生じる。

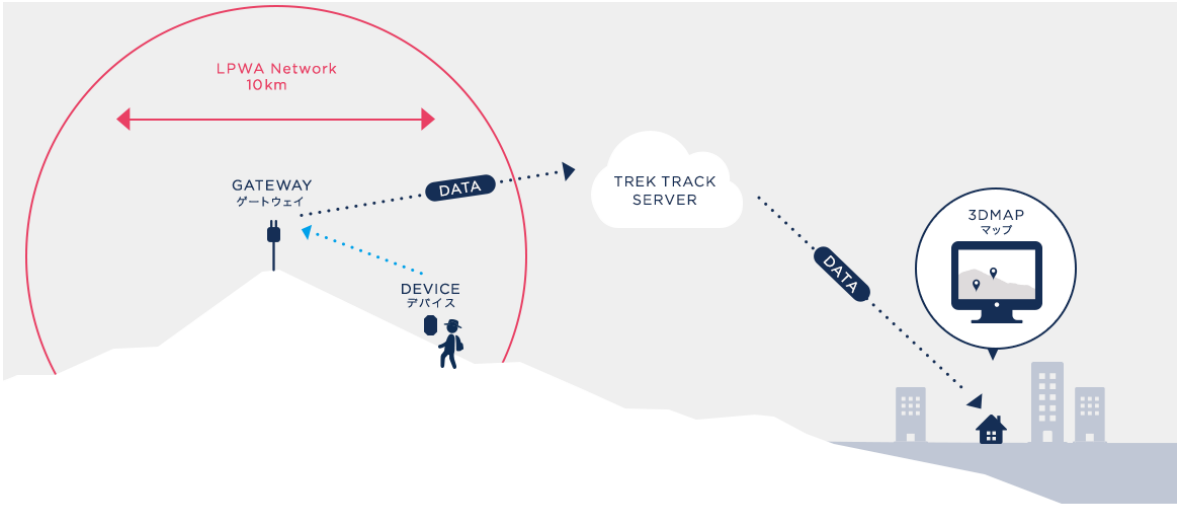

図 6 TREK TRACK のイメージ図

2.3 ジオグラフィカ

2.3.1 特徴

ジオグラフィカ（Geographica）は、山奥などのオフライン環境でも利用できるアプリケーションであり、図 5 のように事前にキャッシュした情報を基に地図表示と GPS ナビゲーションをおこなう。[8]

私物のスマートフォンに無料のアプリケーションをインストールするだけなのでシステム料は発生せず、事前に地図情報を取得していれば全世界中の山岳で利用できる。ほかにも音声によるルート案内をおこなうなど画面操作をせずに情報を得ることができたり GPS ログやマーカーを記録したりと様々な機能が搭載されている。

2.3.2 問題点

多機能である反面、電力消費が激しいためモバイルバッテリーを持ち歩く、平時は機内モードにしておき遭難時でのみ使用するなどの対策が必要であり、操作方法を事前に確認しておかなければならない。また、国土地理院や Google 社の地図情報をインターネット上から取得しているので、仕様や運営方針の変更で使用できなくなる恐れがある。

図 7　ジオグラフィカのイメージ図

第3章 提案手法

2.1.3、2.2.2、2.3.2 で示した問題点を解決するために、登山者が持ち歩いているデバイスから救助要請が発信された際、山岳地帯に散布された GPS 機能を持たない LPWA デバイス同士でマルチホップルーティングをおこないゲートウェイまで信号を繋ぐシステムを提案する。

3.1 着眼点

LPWA は一般的な電池で数年間動作することが長所の 1 つであり、消費電力の大きい GPS 機能を搭載するのは不適当である。しかしアドホックネットワークを利用すれば、通信をおこなえるデバイス同士が近傍にあることがわかるので、相対位置を推察することができる。1 度設置したノードは大きく移動しないため、一部のノードの絶対位置を更新するたびにネットワーク内にある他のノードの位置を推察することができ、それらの情報から救助信号を発したデバイスの位置情報の特定がおこなえる。

アドホックネットワークの核となるマルチホップ通信のルーティング制御は、Wi-SUN では主流の手法であることや LoRa でも可能なことが確認されているため、ネットワーク内にあるデバイスの位置情報を推察する方法を考える。

3.2 提案手法

本研究では、LPWA デバイスの現在位置を推定し、その情報を用いて発信位置からゲートウェイまで確実に信号を繋ぐルートの構築をおこなう。LPWA デバイスは低消費電力で長期的な利用を実現するために自身の位置情報を保持しておらず、発信する情報は自身の固有識別番号と救助信号のみである。LPWA デバイスの位置推定には、パトロール隊員の荷物や野生動物の首輪などに GPS 機能を搭載したものを用い、実際に山中を移動してもらうことで定期的な位置の推定を繰り返しおこなって誤差を縮小する。登山者には救助要請の発信機能が搭載されたクマ避けの鈴などを持ち歩いてもらうことで、山小屋や公衆電話を利用したゲートウェイを介し、電話回線から通報をおこなう。

LPWA デバイスの位置推定のアルゴリズムを図 8、マルチホップルーティングのアルゴリズムを図 9 に示す。

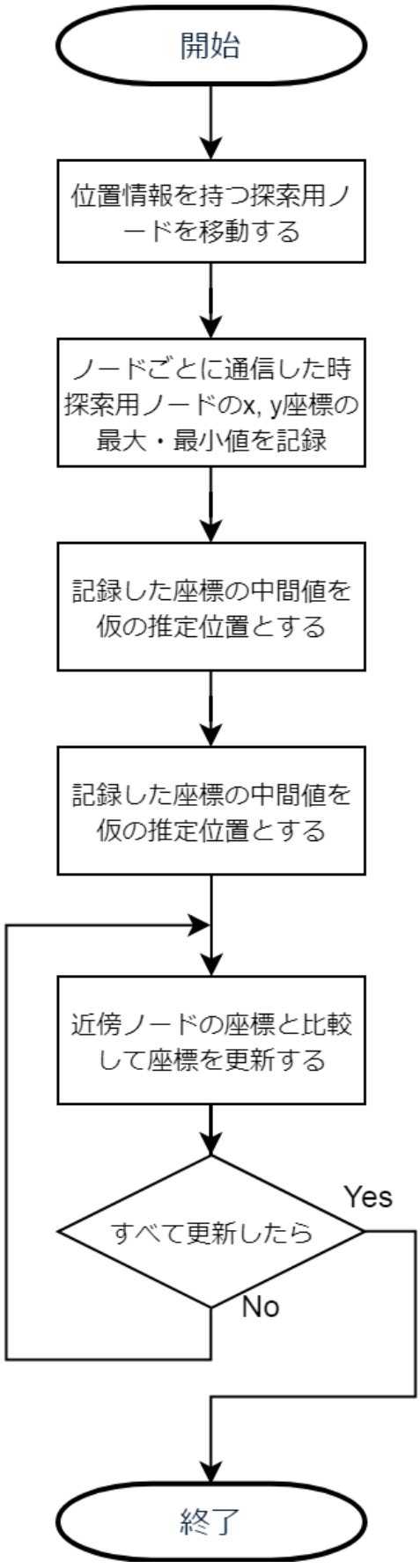

図 8　位置推定アルゴリズム

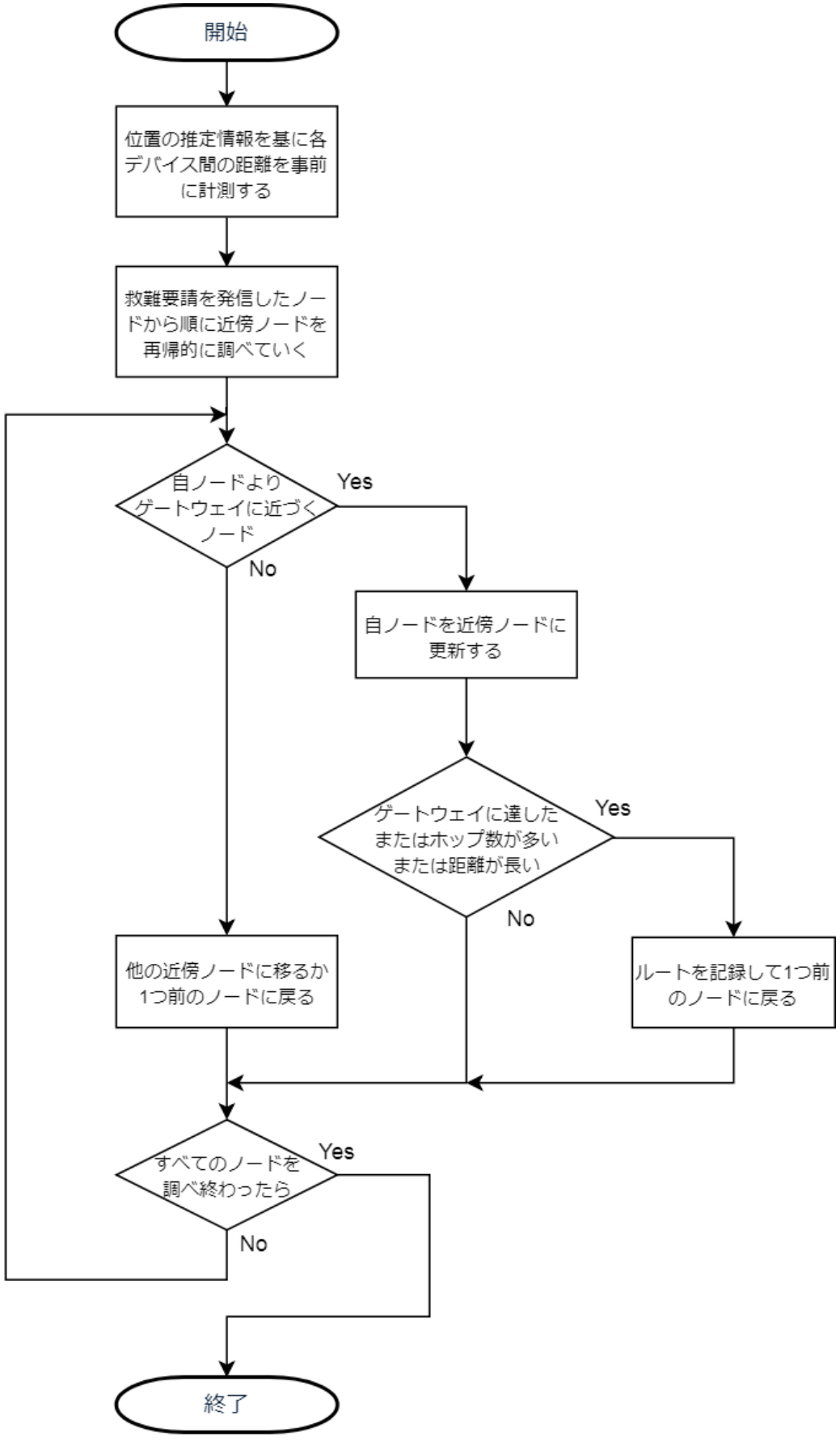

図 9　経路探索アルゴリズム

第4章 実験

第 3 章で提案した位置推定アルゴリズム、経路探索アルゴリズムを実現させるためにそれぞれ 10km 四方の山岳を想定してシミュレーションをおこなう。

4.1 実験環境

本実験にて使用したシミュレーション環境を以下に示す。実装は Python にておこない、乱数生成や計算処理などに Numpy、描画処理に Matplotlib を使用してシミュレーションと評価をおこなった。

- Windows 8.1
- Python 3.7.0
- Numpy 1.14.5
- Matplotlib 2.2.2

4.2 実験 1

実験 1 では、探索用ノードが取得した情報を基にエリア内に存在するノードの位置を推定するアルゴリズムを確立することで、山中にある LPWA デバイス位置の認識を目指す。

4.2.1 実験概要

ランダムで n 地点にノードを配置し、それぞれ実際の位置情報を用いて通信可能距離（tr_d）から近傍ノードを把握する。次に、tr_d*1.0 の円周上を移動する探索ノード A と tr_d*2.0 の円周上を移動する探索ノード B の 2 つを使用して位置を推定する。これにより、A のみが通信した場合は tr_d*1.0 より内側、A と B の両方が通信した場合は tr_d*1.0 から tr_d*2.0、B のみが通信をした場合は tr_d*2.0 から tr_d*3.0、いずれも通信できなかった場合は tr_d*3.0 より外側の範囲内にノードが存在すると推定できる。ここで、電波強度により距離推定が可能と仮定して距離情報を用いた位置推定をおこなう。また、すでに位置が確定しているものとしてゲートウェイの近傍ノードの推察もおこなう。

そのノードが存在すると考えられる範囲を[max_x, min_x], [max_y, min_y]とおき、位置推定のたびに徐々に範囲を狭めていく。この範囲内に近傍ノード以外のノードが存在する場合はお互いの位置を少しずつ離して真値に近づける。そして、近傍ノードの推定位置から一定距離を半径とする円を重ねていき、すべての円が重なる範囲の中心を新たな推定座標として記録する。ただし、近傍ノード以外の周辺ノードの範囲と重なっている場合はその範囲を削る。最後に、推定した位置情報の結果と実際の位置情報を比較して評価をおこなう。

乱数で生成するノード(device_num）の個数を[50, 75, 100, 125, 150]個、通信可能距離を[2.0, 2.5, 3.0, 3.5, 4.0]km の組み合わせで 100 回ずつシミュレーションをおこない評価値を比較する。評価方法は、各ノードの真値と推定値の絶対距離を合計した値(sum_distance)を使用して、100 – (sum_distance/device_num) * 10 [%]から算出して得た値の平均を位置推定率とする。

4.2.2 実験結果

表 4 に算出した位置推定率の結果を示す。なお、横軸はノード数、縦軸は通信可能距離を表している。また、ノード数が 100、通信可能距離が 3.0km を想定したシミュレーションで使用した実際の位置情報と推定した位置情報の例をそれぞれ図 10、図 11 に示す。

表 4 ノード数と通信可能距離ごとの位置推定率

	50	75	100	125	150
2.0	38.535	40.341	47.903	52.741	56.732
2.5	58.987	63.765	70.834	76.867	80.083
3.0	77.644	82.362	86.137	89.213	89.343
3.5	89.346	91.739	92.660	92.949	94.024
4.0	92.381	93.716	94.030	94.301	94.327

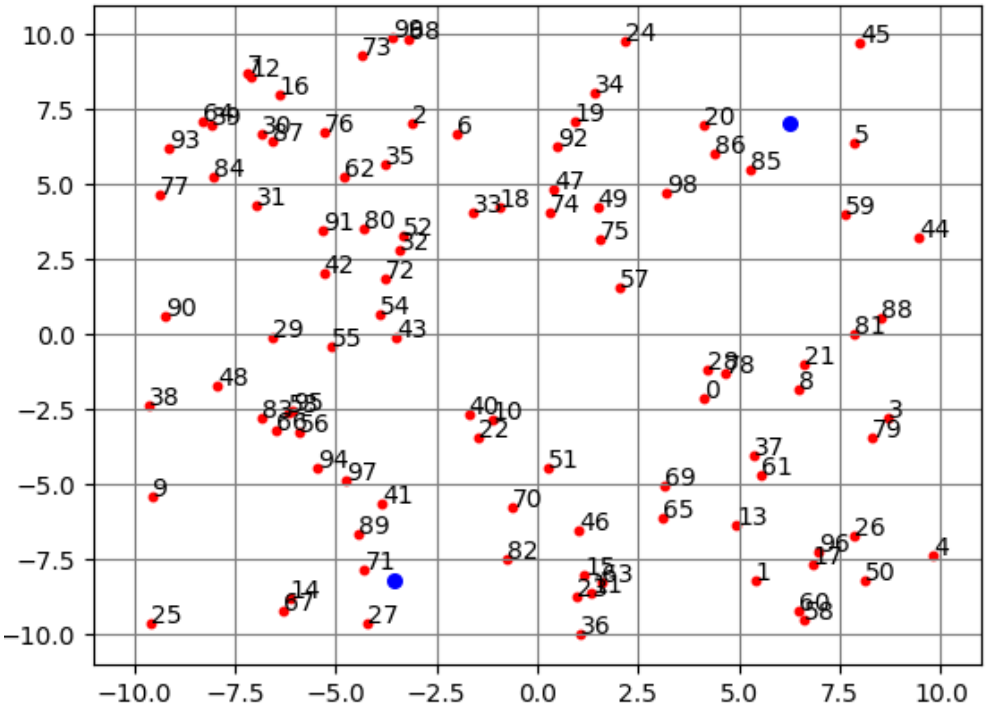

図 10 実際のノード位置情報

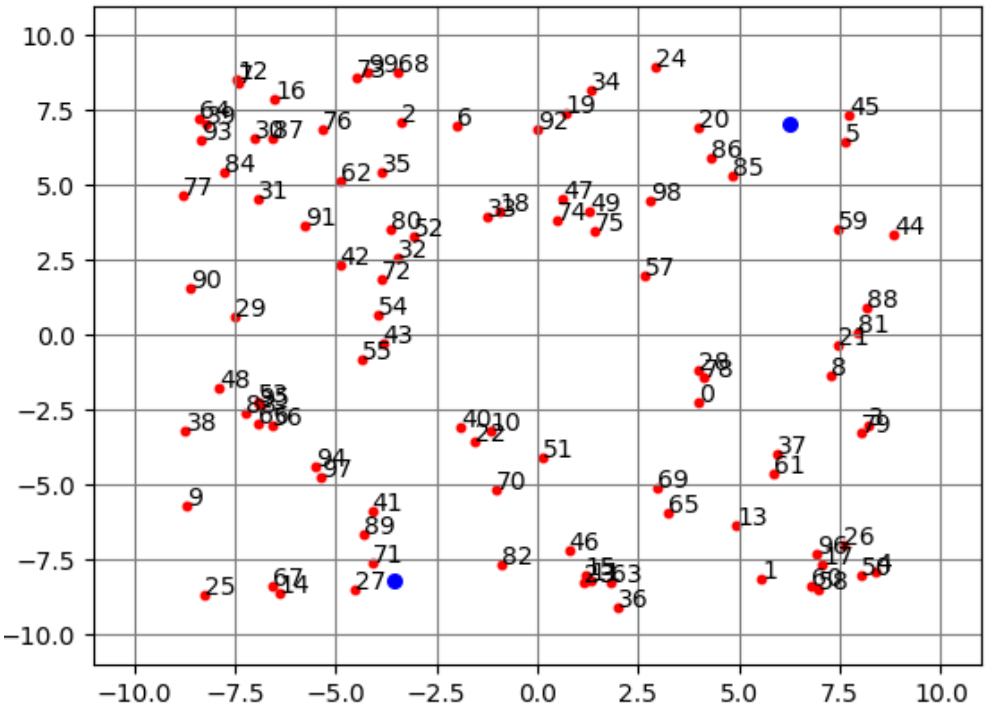

図 11 推定したノード位置情報

4.3 実験 2

実験 2 では、推定位置情報を持つ全ノードを用いて発信位置からゲートウェイまでの最適ルートを導出するアルゴリズムを確立と発信位置の特定をおこなうことで、救助要請の実現を目指す。

4.3.1 実験概要

実験 1 で作成した推定位置情報を使用して実験 2 を進めていく。

まず、推定した発信位置からゲートウェイの位置までの直線距離を計算し、もしルートが繋がった場合はそのままルート構築を終了する。ルートが繋がらなかった場合は各ノードの通信可能圏内にある近傍ノードの距離をすべて計算し、ゲートウェイのある方向へ続くノードのみを記録する。

取得した各ノードの距離情報を用いてルートを再帰的に構築していく。あまりにホップ数が増えたり通信距離が長くなったりすると、通信が失敗する可能性が高くなることや通信可能ルートが増えすぎて断定に時間がかかりすぎるなどの懸念があるため、推定したルートの最小ホップ数と最短通信距離を基準に選別していく。推定されたルートの中から、直線距離とホップ数*通信可能距離*0.1 を足した値が最も小さいものを最適ルートとする。

実験 1 と同様に、乱数で生成するノードの個数を[50, 75, 100, 125, 150]個、通信可能距離を[2.0, 2.5, 3.0, 3.5, 4.0]km の組み合わせで 100 回ずつシミュレーションをおこない、発信位置からゲートウェイまでのルートが構築できた回数を比較する。ただし、実験のために発信位置とゲートウェイはあらかじめ離れた範囲同士でランダムに配置する。また、救助信号を受信できた推定ノードの中心地点を LPWA デバイスの発信位置とし、真値との絶対距離の平均値で評価する。

4.3.2 実験結果

表 5 にルートを構築した回数、表 6 に推定した発信位置の平均誤差を示す。なお、横軸はノード数、縦軸は通信可能距離を表している。また、図 11 で推定した位置情報をもとに構築した最適ルートの例を図 12 に示す。

表 5 ノード数と通信可能距離ごとのルート構築回数（100 回）

	50	75	100	125	150
2.0	0	0	3	6	32
2.5	8	23	37	52	69
3.0	13	64	83	98	99
3.5	57	91	93	99	100
4.0	89	98	97	100	100

表 6 ノード数と通信可能距離ごとの推定発信位置平均誤差[km]

	50	75	100	125	150
2.0	6.375	6.379	6.475	7.396	3.997
2.5	2.298	3.025	2.122	3.348	2.531
3.0	2.100	1.296	1.258	1.061	0.815
3.5	1.820	1.609	0.920	0.703	0.669
4.0	1.108	1.436	1.044	0.908	0.636

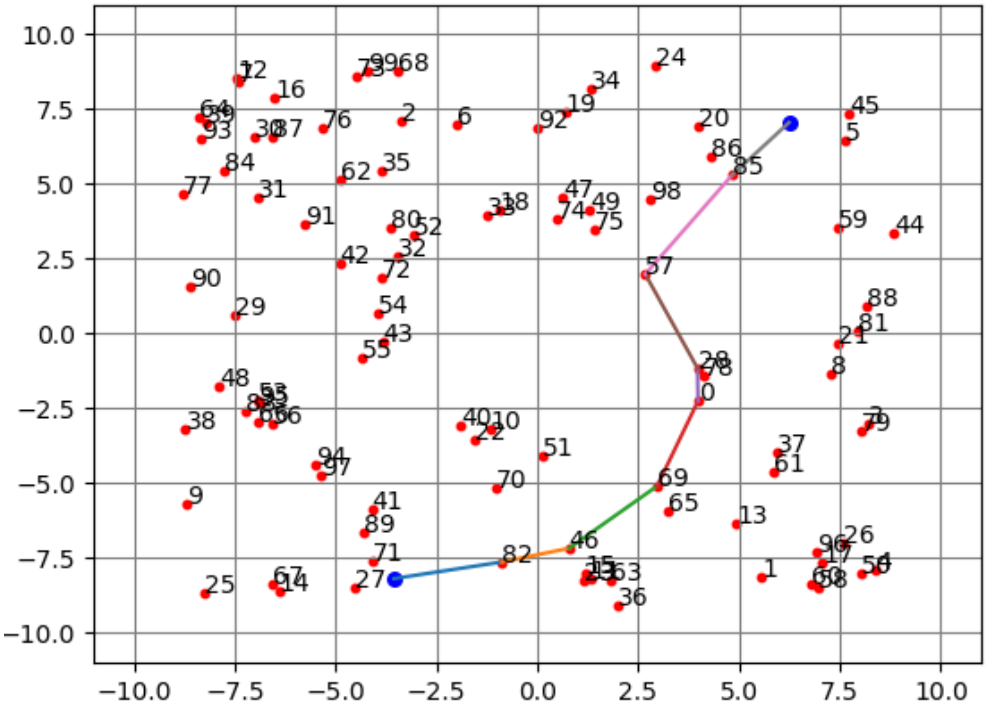

図 12 構築した最適ルート

4.4 考察

実験1の探索用ノードは2つの円の円周上を移動して原点から通信可能距離*3.0までカバーしているため、3.0km 以上の通信可能距離があれば高い推定率を得ることができた。この要因として、ノードとの距離情報だけでは探索用ノードの進行方向から見てどちらにあるか推定するのは難しいが、一定範囲を 2 つの探索ノードが取得することで方向の特定をおこなえたことが挙げられる。一方で手法の特性上、原点から離れた最も外側の範囲での推定は不十分になるため、通信可能距離が短いほど位置推定率が低くなった。また、ノード個数が多いほど推定回数が多くなるため位置推定率が高くなった。

実験 2 ではルート構築をおこなったが、シミュレーションに用いたノード個数では少なくとも 3.0km 以上の通信可能距離が必要であることがわかった。また、発信位置を推定する際、近傍ノードの中心位置から特定しているため近傍ノードに囲まれていないと大きな誤差が生じてしまった。この結果からもシステム構築には一定のノード数を設置しなければならないと考えられる。

本シミュレーションは一切の位置情報を知らない状態で 1 度のみ探索用ノードを移動させてシミュレーションをおこなったが、実際の利用シーンではあらかじめ設置場所の特定ができていることや幾度となく探索用ノードによる距離推定がおこなわれることが想定されるため、より正確な座標を記録できると考えられる。また、10km 四方の山岳にランダムでノードを配置した状況でシミュレーションしているため、より狭い範囲での利用や一定間隔を保ちながらノードを設置した場合を考慮するとルート構築や位置推定に必要なノード数は実験結果よりも少なくなると考えられる。

以上より、通信可能距離が 3.0km 以上あれば 75 個程度の設置ノード数で十分推定が可能といえる。ただし、山岳地帯で現在の LPWA 技術を利用することを考えると通信可能距離はせいぜい 2.0km 程度だと推察できるので、確実なルート構築と救助要請の発信位置の特定のためにより多量のノードを設置すべきである。

第5章 まとめ

本研究では、LPWA 技術を用いて山岳地帯全体にネットワーク環境を構築して山岳遭難事故の発生を防止するための事前準備として、LPWA 機器の位置推定と救助信号の発信位置からゲートウェイまでのルートを構築するシミュレーション実験をおこなった。

実験 1、実験 2 から各ノードの通信可能距離が 3.5 以上ある場合、10km 四方の山岳では 75 個以上のノードがあれば十分に位置推定をおこないルートを構築できることがわかった。

現在の LPWA 技術では障害物の影響を強く受けるため信号強度による距離推定は難しいが、昨今の技術進化スピードを鑑みると近い将来に可能になるだろう。また、今回は探索用ノードが円周上を移動しているため高い精度で位置を推定できているが、現実の地形には高さ（z 軸）や障害物、山道といった多くの情報が存在している。より実用的なシステムにするために、3 次元地形データなどから多くの情報を絡めて実際の状況を再現したシミュレーション実験をおこなうことが今後の課題として挙げられる。

謝辞

本研究を進めるにあたりご指導、ご助言を頂きました三好　力教授に深く感謝いたします。また、日常の議論を通じて多くの知識や示唆を頂いた三好研究室の皆様に感謝します。

参考文献

[1]“2017 年の国内アウトドア市場は前年比 103.2%、今後も好調に推移する見込み”－@DIME（https://dime.jp/genre/582318/）

[2]“平成 29 年における山岳遭難の概況”－ 警察庁生活安全局地域課(2018 年 6 月 21 日)

[3]“IoT や M2M で注目される無線通信技術「LPWA」とは”- Tech Factory (http://techfactory.itmedia.co.jp/tf/articles/1701/30/news008.html)

[4]“平成 29 年版 情報通信白書”- 総務省 (http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h29/html/nc133220.html)

[5]“遅いけど遠くまで届く LPWA IoT 無線通信の本命か”－ 日経 NETWORK(2017 年 1 月号)

[6]“LPWA の最新動向と今後の展望”- 千葉大学(2018 年 6 月 6 日)

[7] TREK TRACK（https://trektrack.jp）

[8] ジオグラフィカ（http://geographica.biz）

[9]“LoRa による無線マルチホップネットワークの構築”- 大阪工業大学(2017 年 11 月 16 日)

付録

Position_Estimation.py

```
import matplotlib.pyplot as plt
import numpy as np
import math
# マルチホップ通信可能距離(transmission_distance)
tr_d = 4.0
# 設置する機器数
device_num = 100
# マップ範囲
map_width = 20
map_height = 20
# ノード位置
np.set_printoptions(precision=3, floatmode='fixed')
x = np.random.uniform(-map_width/2, map_width/2, device_num)
y = np.random.uniform(-map_height/2, map_height/2, device_num)
signal_position = {'x':np.random.uniform(-map_width/2, 0),
                   'y':np.random.uniform(-map_height/2, 0)}
gateway_position = {'x':np.random.uniform(0, map_width/2),
                    'y':np.random.uniform(0, map_height/2)}
# fig = plt.figure(figsize=(14,5)) # 先に figure を作っておく
def Display_Position(estimate):
    # グラフ設定
    # plt.subplot(1, 2, 1)
    plt.cla()
    plt.xlim([-map_width/2-1, map_width/2+1])
    plt.ylim([-map_height/2-1, map_height/2+1])
    plt.grid(color='gray')
    # 全ノードの位置をプロット
    plt.scatter(signal_position['x'], signal_position['y'], marker="o", s=30, c="blue")
    plt.scatter(gateway_position['x'], gateway_position['y'], marker="o", s=30, c="blue")
    plt.scatter(x, y, marker="o", s=10, c="red")
    for i,(a,b) in enumerate(zip(x, y)):
        plt.annotate(str(i),(a,b))  # ラベル
    plt.show()
    # グラフ設定
    # plt.subplot(1, 2, 2)
    plt.cla()
    plt.xlim([-(map_width/2)-1, map_width/2+1])
    plt.ylim([-(map_height/2)-1, map_height/2+1])
    plt.grid(color='gray')
    plt.scatter(signal_position['x'], signal_position['y'], marker="o", s=30, c="blue")
    plt.scatter(gateway_position['x'], gateway_position['y'], marker="o", s=30, c="blue")
    for k, v in estimate.items():
        plt.scatter(v['x'], v['y'], marker="o", s=10, c="red")
        plt.annotate(k,(v['x'], v['y']))  # ラベル
    plt.show()
def calc_distance(x1, y1, x2, y2):
    a = (x2-x1)**2
    b = (y2-y1)**2
    c = round(math.sqrt(a+b), 3)
    return c
# 近隣ノード位置の距離
neighborhood = {}
def Neighborhood_Position():
    # 一ノードの近隣ノードの検出
    for i in range(0, device_num):
        level = []
        # 全ノードとの比較
        for j in range(0, device_num):
            # 自ノードは省く
            if j == i:
                continue
            d = calc_distance(x[i], y[i], x[j], y[j])
            if d < tr_d:
                level.append(str(j))
        # signal との比較
        d = calc_distance(x[i], y[i], signal_position['x'], signal_position['y'])
        if d < tr_d:
            level.append('signal')
        # gateway との比較
        d = calc_distance(x[i], y[i], gateway_position['x'], gateway_position['y'])
        if d < tr_d:
            level.append('gateway')
        neighborhood.update({str(i):level})
    # signal の近隣ノードの検出
    signal = []
    for k in range(0, device_num):
        d = calc_distance(signal_position['x'], signal_position['y'], x[k], y[k])
        if d < tr_d:
            signal.append(str(k))
    d = calc_distance(signal_position['x'], signal_position['y'], gateway_position['x'], gateway_position['y'])
    if d < tr_d:
        level.append('gateway')
    neighborhood.update({'signal':signal})
    # gateway の近隣ノードの検出
    gateway = []
    for k in range(0, device_num):
        d = calc_distance(gateway_position['x'], gateway_position['y'], x[k], y[k])
        if d < tr_d:
            gateway.append(str(k))
    d = calc_distance(gateway_position['x'], gateway_position['y'], signal_position['x'], signal_position['y'])
    if d < tr_d:
        level.append('signal')
    neighborhood.update({'gateway':gateway})
def Evaluation(estimate):
    sum_distance = 0
    for i in range(0, device_num):
        sum_distance += calc_distance(estimate[str(i)]['x'], estimate[str(i)]['y'], x[i], y[i])
    print(round(100-(sum_distance/device_num)*10, 3), '%')
```

Route_Search.py

```
import matplotlib.pyplot as plt
import numpy as np
import math
import copy
import sys
import Position_Estimation as PE
transmission_distance = PE.tr_d
device_num = PE.device_num
map_width = PE.map_width
map_height = PE.map_height
x = []
y = []
signal_position = {}
gateway_position = {}
straight_distance = 999
def Actual_Position():
    # グラフ設定
    plt.cla()
    plt.xlim([-11, 11])
    plt.ylim([-11, 11])
    plt.grid(color='gray')
    # ノードと救助信号・ゲートウェイの位置をプロット
    plt.scatter(signal_position['x'], signal_position['y'], marker="o", s=30, c="blue")
    plt.scatter(gateway_position['x'], gateway_position['y'], marker="o", s=30, c="blue")
    plt.scatter(x, y, marker="o", s=10, c="red")
    for i,(a,b) in enumerate(zip(x, y)):
        plt.annotate(str(i),(a,b))  # ラベル
def calc_distance(x1, y1, x2, y2):
    a = (x2-x1)**2
    b = (y2-y1)**2
    c = round(math.sqrt(a+b), 3)
    return c
# 各ノード間の距離計測
distance = {}
def Measure_Distance():
    start = {}
    # 1 発で繋げるか確認
    if straight_distance < transmission_distance:
        start.update({'goal':straight_distance})
    else:
        for i in range(0, device_num):
            if x[i]==0.0 and y[i]==0.0:
                continue
            level = {}
            # 発信源の位置に近いところ
            start_distance = calc_distance(signal_position['x'], signal_position['y'], x[i], y[i])
            if start_distance < transmission_distance:
                start.update({str(i):start_distance})
            # ゲートウェイの位置に近いところ
            base_goal_distance = calc_distance(gateway_position['x'], gateway_position['y'], x[i], y[i])
            if base_goal_distance < transmission_distance:
                level.update({'goal':base_goal_distance})
            # ゲートウェイの位置に近くない場合
            else:
                for j in range(0, device_num):
                    if x[j]==0.0 and y[j]==0.0:
                        continue
                    d = calc_distance(x[i], y[i], x[j], y[j])
                    goal_distance = calc_distance(gateway_position['x'], gateway_position['y'], x[j], y[j])
                    # 前よりもゴールに近づいているノードに絞る
                    if d < transmission_distance and transmission_distance*0.1 < d and goal_distance < base_goal_distance:
                        level.update({str(j):d})
                distance.update({str(i):level})
    distance.update({'start':start})
# ルート探索
route = {}
temp_r = ['start']
temp_d = [0.0]
route_count = 0  # グローバル変数
# point: 指定された key 名
aim_count = 99
aim_distance = 999.999
def Route_Search(point):
    global aim_count
    global aim_distance
    global route_count
    for k, v in distance[point].items():
        temp_r.append(k)
        temp_d.append(v)
        # あまりにホップ数が多くなったら
        # あまりに距離が長くなったら
        if len(temp_r)-1 > aim_count or sum(temp_d) > aim_distance:
            temp_r.pop()
            temp_d.pop()
        # ゴールまでたどり着いたら
        elif k == 'goal':
            temp = {}
            r = copy.deepcopy(temp_r)
            d = copy.deepcopy(temp_d)
            temp.update({"r":r, "d":d})
            aim_count = len(r) if len(r) < aim_count else aim_count
            aim_distance = round(sum(d), 3) if sum(d) < aim_distance else aim_distance
            route.update({str(route_count):temp})
            route_count += 1
            temp_r.pop()
            temp_d.pop()
        else:
            Route_Search(k)
    if temp_r and temp_d:
        temp_r.pop()
        temp_d.pop()
# 経路の描画
shortest_distance = 999.999 # 最短距離
def Plot_Route():
    global shortest_distance
    best_route = -1
    # 最短経路の算出
    for k, v in route.items():
        temp_dis = round(sum(v['d']), 3)
        temp_dis += len(v['r'])*transmission_distance*0.1
        if temp_dis < shortest_distance:
            best_route = v['r']
            shortest_distance = temp_dis
    # 最短経路の描画
    if best_route != -1:
        for i in range(0, len(best_route)-1):
            if best_route[i] == 'start' and best_route[i+1] == 'goal':
```

```
                    plt.plot([signal_position['x'],
gateway_position['x']],                    [signal_position['y'],
gateway_position['y']])
                elif best_route[i] == 'start':
                    b = int(best_route[i+1])
                    plt.plot([signal_position['x'],          x[b]],
[signal_position['y'], y[b]])
                elif best_route[i+1] == 'goal':
                    a = int(best_route[i])
                    plt.plot([x[a],       gateway_position['x']],
[y[a], gateway_position['y']])
                else:
                    a = int(best_route[i])
                    b = int(best_route[i+1])
                    plt.plot([x[a], x[b]], [y[a], y[b]])
    print(best_route)
def Signal_Estimation():
    global signal_position
    signal_x = 0
    signal_y = 0
    length = len(PE.neighborhood['signal'])
    if length > 0:
        for v in PE.neighborhood['signal']:
            if v == 'gateway':
                signal_x += gateway_position['x']
                signal_y += gateway_position['y']
            else:
                signal_x += x[int(v)]
                signal_y += y[int(v)]
        signal_x = round(signal_x/length, 3)
        signal_y = round(signal_y/length, 3)
    # print('calc', signal_x, signal_y)
    #       print('real',       round(signal_position['x'],       3),
round(signal_position['y'], 3))
    print('error',    round(calc_distance(signal_x,    signal_y,
signal_position['x'], signal_position['y']), 3))
    signal_position['x'] = signal_x
    signal_position['y'] = signal_y
def main(estimate, signal, gateway):
    global x
    global y
    global signal_position
    global gateway_position
    global straight_distance
    global aim_count
    global aim_distance
    for k, v in estimate.items():
        x.append(v['x'])
        y.append(v['y'])
    gateway_position['x'] = gateway['x']
    gateway_position['y'] = gateway['y']
    signal_position['x'] = signal['x']
    signal_position['y'] = signal['y']
    straight_distance   =   calc_distance(signal_position['x'],
signal_position['y'],                    gateway_position['x'],
gateway_position['y'])
    aim_count                                                    =
math.ceil(straight_distance*2.5/transmission_distance)
    aim_distance = round(straight_distance*2.5, 3)
    Actual_Position()
    Measure_Distance()
    Route_Search('start')
    Plot_Route()
    Signal_Estimation()
    # print('route_count : ' + str(route_count))
    # print('straight_distance : ' + str(straight_distance))
    # print('shortest_distance : ' + str(shortest_distance) +
' (+' + str(round(shortest_distance-straight_distance, 3)) +
')')
    plt.show()
```

PE_method.py

```
import matplotlib.pyplot as plt
import numpy as np
import math
import copy
import sys
import Position_Estimation as PE
import Route_Search as RS
tr_d = PE.tr_d
device_num = PE.device_num
map_width = PE.map_width
map_height = PE.map_height
x = PE.x
y = PE.y
signal_position = PE.signal_position
gateway_position = PE.gateway_position
# 探索用変数
plot_intervel = [0.10, 0.25] # 探索する距離間隔
estimate = {}
def presume_position(estimate_array):
    # 初期化
    expect_position = {}    # ノードの分布予想円(max_x, max_y, min_x, min_y)
    exclusion_range = {}
    update_flag = {}        # 情報更新できそうなノードをフラグ管理
    count = 0
    for i in range(0, device_num):
        exclusion_range[str(i)] = {}
        update_flag[str(i)] = 0
    # gateway が観測できるノード
    print('gateway', PE.neighborhood['gateway'])
    for v in PE.neighborhood['gateway']:
        distance = PE.calc_distance(gateway_position['x'], gateway_position['y'], x[int(v)], y[int(v)]) * 1.5
        position = {'max_x':round(gateway_position['x']+distance, 3) if gateway_position['x']+distance < map_width/2 else map_width/2,

'min_x':round(gateway_position['x']-distance, 3) if gateway_position['x']-distance > -map_width/2 else -map_width/2,

'max_y':round(gateway_position['y']+distance, 3) if gateway_position['y']+distance < map_height/2 else map_height/2,

'min_y':round(gateway_position['y']-distance, 3) if gateway_position['y']-distance > -map_height/2 else -map_height/2}
        expect_position.update({v:position})
    # transmit_node が観測したノード
    for k, v in estimate_array.items():
        position = {}
        # 探索ノードからの距離
        xd = v['calc_x']
        yd = v['calc_y']
        correct = tr_d*0.5
        position = {'max_x':round(xd+correct, 3) if abs(xd+correct) < map_width/2 else map_width/2,
                    'min_x':round(xd-correct, 3) if abs(xd-correct) < map_width/2 else -map_width/2,
                    'max_y':round(yd+correct, 3) if abs(yd+correct) < map_height/2 else map_height/2,
                    'min_y':round(yd-correct, 3) if abs(yd-correct) < map_height/2 else -map_height/2}
        if xd+correct>10 or yd+correct>10 or xd-correct<-10 or yd-correct<-10:
            print(xd, yd, correct, position)
        expect_position.update({k:position})
        update_flag[k] = 1
    # # あらかじめ場所を知っているノードを記録
    # for i in range(0, device_num):
    #     if i%5==0:
    #         position = {'max_x':round(x[i], 3),
    #                     'min_x':round(x[i], 3),
    #                     'max_y':round(y[i], 3),
    #                     'min_y':round(y[i], 3)}
    #         expect_position.update({str(i):position})
    #         update_flag[str(i)] = 1
    # 近傍ノードの確認
    for i in range(0, device_num):
        print(i, PE.neighborhood[str(i)])
    # 観測できなかったノードを近隣ノードから推察
    for i in range(0, device_num):
        if not str(i) in expect_position.keys():
            length = 0
            max_x = 0
            min_x = 0
            max_y = 0
            min_y = 0
            for v in PE.neighborhood[str(i)]:
                if v=='signal' or v=='gateway':
                    continue
                if v in expect_position.keys():
                    max_x += expect_position[v]['max_x']
                    min_x += expect_position[v]['min_x']
                    max_y += expect_position[v]['max_y']
                    min_y += expect_position[v]['min_y']
                    length += 1
            if length>1:
                max_x = max_x/length if abs(max_x/length) < map_width/2 else map_width/2
                min_x = min_x/length if abs(min_x/length) < map_width/2 else -map_width/2
                max_y = max_y/length if abs(max_y/length) < map_height/2 else map_height/2
                min_y = min_y/length if abs(min_y/length) < map_height/2 else -map_height/2
                position = {'max_x':round(max_x + tr_d if max_x >= 0 else max_x, 3),
                            'min_x':round(min_x - tr_d if min_x < 0 else min_x, 3),
                            'max_y':round(max_y + tr_d if max_y >= 0 else max_y, 3),
                            'min_y':round(min_y - tr_d if min_y < 0 else min_y, 3)}
                expect_position.update({str(i):position})
                update_flag[str(i)] = 1
                # print(str(i), position, round(x[i], 3), round(y[i], 3))
    # 近隣ノード以外と近い場合は場所を離す
    for n in range(20):
        for i in range(0, device_num):
            for j in range(0, device_num):
                if i!=j and str(i) in expect_position.keys() and str(j) in expect_position.keys() and not str(j) in PE.neighborhood[str(i)]:
                    max_xi = expect_position[str(i)]['max_x']
```

```
                        min_xi = expect_position[str(i)]['min_x']
                        max_yi = expect_position[str(i)]['max_y']
                        min_yi = expect_position[str(i)]['min_y']
                        max_xj = expect_position[str(j)]['max_x']
                        min_xj = expect_position[str(j)]['min_x']
                        max_yj = expect_position[str(j)]['max_y']
                        min_yj = expect_position[str(j)]['min_y']
                        # 正確性に欠けるノードは利用しない
                        if abs(max_xj-(max_xj+min_xj)/2) < tr_d and abs(min_xj-(max_xj+min_xj)/2) < tr_d ¥
                            and abs(max_yj-(max_yj+min_yj)/2) < tr_d and abs(min_yj-(max_yj+min_yj)/2) < tr_d:
                            if min_xi < max_xj and max_xj < max_xi and min_yi < (max_yj+min_yj)/2 and (max_yj+min_yj)/2 < max_yi:
                                min_xi = expect_position[str(i)]['min_x'] + tr_d*0.1
                                max_xj = expect_position[str(j)]['max_x'] - tr_d*0.1
                            if min_xi < min_xj and min_xj < max_xi and min_yi < (max_yj+min_yj)/2 and (max_yj+min_yj)/2 < max_yi:
                                max_xi = expect_position[str(i)]['max_x'] - tr_d*0.1
                                min_xj = expect_position[str(j)]['min_x'] + tr_d*0.1
                            if min_yi < max_yj and max_yj < max_yi and min_xi < (max_xj+min_xj)/2 and (max_xj+min_xj)/2 < max_xi:
                                min_yi = expect_position[str(i)]['min_y'] + tr_d*0.1
                                max_yj = expect_position[str(j)]['max_y'] - tr_d*0.1
                            if min_yi < min_yj and min_yj < max_yi and min_xi < (max_xj+min_xj)/2 and (max_xj+min_xj)/2 < max_xi:
                                max_yi = expect_position[str(i)]['max_y'] - tr_d*0.1
                                min_yj = expect_position[str(j)]['min_y'] + tr_d*0.1
                            position = {'max_x':round(max_xi, 3) if abs(max_xi) < map_width/2 else map_width/2,

'min_x':round(min_xi, 3) if abs(min_xi) < map_width/2 else -map_width/2,

'max_y':round(max_yi, 3) if abs(max_yi) < map_height/2 else map_height/2,

'min_y':round(min_yi, 3) if abs(min_yi) < map_height/2 else -map_height/2}

expect_position.update({str(i):position})
                            update_flag[str(i)] = 1
                            position = {'max_x':round(max_xj, 3) if abs(max_xj) < map_width/2 else map_width/2,

'min_x':round(min_xj, 3) if abs(min_xj) < map_width/2 else -map_width/2,

'max_y':round(max_yj, 3) if abs(max_yj) < map_height/2 else map_height/2,

'min_y':round(min_yj, 3) if abs(min_yj) < map_height/2 else -map_height/2}

expect_position.update({str(j):position})
                            update_flag[str(j)] = 1
    # 位置推定
    while 1 in update_flag.values():
        for i in range(0, device_num):
            position = {}
            if str(i) in expect_position.keys():
                max_x = (expect_position[str(i)]['max_x']+expect_position[str(i)]['min_x'])/2
                min_x = (expect_position[str(i)]['max_x']+expect_position[str(i)]['min_x'])/2
                max_y = (expect_position[str(i)]['max_y']+expect_position[str(i)]['min_y'])/2
                min_y = (expect_position[str(i)]['max_y']+expect_position[str(i)]['min_y'])/2
                length = 1
                # 十分に範囲を推定しているため continue
                if abs(max_x-min_x) < 1.0 and abs(max_y-min_y) < 1.0:
                    update_flag[str(i)] = 0
                    continue
                max_x = max_x + tr_d*0.7 if abs(max_x+tr_d*0.7) < map_width/2 else map_width/2
                min_x = min_x - tr_d*0.7 if abs(min_x-tr_d*0.7) < map_width/2 else -map_width/2
                max_y = max_y + tr_d*0.7 if abs(max_y+tr_d*0.7) < map_height/2 else map_height/2
                min_y = min_y - tr_d*0.7 if abs(min_y-tr_d*0.7) < map_height/2 else -map_height/2
            else:
                max_x = 0
                min_x = 0
                max_y = 0
                min_y = 0
                length = 0
            for v in PE.neighborhood[str(i)]:
                if v=='signal' or v=='gateway':
                    continue
                if v in expect_position.keys():
                    max_x += (expect_position[v]['max_x']+expect_position[v]['min_x'])/2 + tr_d*0.7
                    min_x += (expect_position[v]['max_x']+expect_position[v]['min_x'])/2 - tr_d*0.7
                    max_y += (expect_position[v]['max_y']+expect_position[v]['min_y'])/2 + tr_d*0.7
                    min_y += (expect_position[v]['max_y']+expect_position[v]['min_y'])/2 - tr_d*0.7
                    # max_x = max_x if abs(max_x) <
```

```
map_width/2 else map_width/2
                        # min_x = min_x if abs(min_x) < map_width/2 else -map_width/2
                        # max_y = max_y if abs(max_y) < map_height/2 else map_height/2
                        # min_y = min_y if abs(min_y) < map_height/2 else -map_height/2
                        length += 1
                if length > 0:
                    max_xi = max_x/length
                    min_xi = min_x/length
                    max_yi = max_y/length
                    min_yi = min_y/length
                    for jk, jv in expect_position.items():
                        # 近傍ノード以外から範囲を絞る
                        if not jk in PE.neighborhood[v]:
                            max_xj = jv['max_x']*1.1
                            min_xj = jv['min_x']*1.1
                            max_yj = jv['max_y']*1.1
                            min_yj = jv['min_y']*1.1
                            if abs(max_xi-max_xj) < tr_d/2 and abs(min_xi-min_xj) < tr_d/2 and abs(max_yi-max_yj) < tr_d/2 and abs(min_yi-min_yj) < tr_d/2:
                                if min_xi < max_xj and max_xj < max_xi and min_yi < (max_yj+min_yj)/2 and (max_yj+min_yj)/2 < max_yi:
                                    min_xi = max_xj
                                if min_xi < min_xj and min_xj < max_xi and min_yi < (max_yj+min_yj)/2 and (max_yj+min_yj)/2 < max_yi:
                                    max_xi = min_xj
                                if min_yi < max_yj and max_yj < max_yi and min_xi < (max_xj+min_xj)/2 and (max_xj+min_xj)/2 < max_xi:
                                    min_yi = max_yj
                                if min_yi < min_yj and min_yj < max_yi and min_xi < (max_xj+min_xj)/2 and (max_xj+min_xj)/2 < max_xi:
                                    max_yi = min_yj
                            position = {'max_x':round(max_xi, 3) if abs(max_xi) < map_width/2 else map_width/2,
                                        'min_x':round(min_xi, 3) if abs(min_xi) < map_width/2 else -map_width/2,
                                        'max_y':round(max_yi, 3) if abs(max_yi) < map_height/2 else map_height/2,
                                        'min_y':round(min_yi, 3) if abs(min_yi) < map_height/2 else -map_height/2}
                            if not str(i) in expect_position.keys() or not expect_position[str(i)] == position:

expect_position.update({str(i):position})
                                update_flag[str(i)] = 1
                                count += 1
                            else:
                                update_flag[str(i)] = 0
        # if count >= 100000:
        #     break
    # 近隣ノード以外と近い場合は場所を離す
    for n in range(20):
        for i in range(0, device_num):
            for j in range(0, device_num):
                if i!=j and str(i) in expect_position.keys() and str(j) in expect_position.keys() and not str(j) in PE.neighborhood[str(i)]:
                    max_xi = expect_position[str(i)]['max_x']
                    min_xi = expect_position[str(i)]['min_x']
                    max_yi = expect_position[str(i)]['max_y']
                    min_yi = expect_position[str(i)]['min_y']
                    max_xj = expect_position[str(j)]['max_x']
                    min_xj = expect_position[str(j)]['min_x']
                    max_yj = expect_position[str(j)]['max_y']
                    min_yj = expect_position[str(j)]['min_y']
                    # 正確性に欠けるノードは利用しない
                    if abs(max_xj-(max_xj+min_xj)/2) < tr_d and abs(min_xj-(max_xj+min_xj)/2) < tr_d ¥
                            and abs(max_yj-(max_yj+min_yj)/2) < tr_d and abs(min_yj-(max_yj+min_yj)/2) < tr_d:
                        if min_xi < max_xj and max_xj < max_xi and min_yi < (max_yj+min_yj)/2 and (max_yj+min_yj)/2 < max_yi:
                            min_xi = expect_position[str(i)]['min_x'] + tr_d*0.1
                            max_xj = expect_position[str(j)]['max_x'] - tr_d*0.1
                        if min_xi < min_xj and min_xj < max_xi and min_yi < (max_yj+min_yj)/2 and (max_yj+min_yj)/2 < max_yi:
                            max_xi = expect_position[str(i)]['max_x'] - tr_d*0.1
                            min_xj = expect_position[str(j)]['min_x'] + tr_d*0.1
                        if min_yi < max_yj and max_yj < max_yi and min_xi < (max_xj+min_xj)/2 and (max_xj+min_xj)/2 < max_xi:
                            min_yi = expect_position[str(i)]['min_y'] + tr_d*0.1
                            max_yj = expect_position[str(j)]['max_y'] - tr_d*0.1
                        if min_yi < min_yj and min_yj < max_yi and min_xi < (max_xj+min_xj)/2 and (max_xj+min_xj)/2 < max_xi:
                            max_yi = expect_position[str(i)]['max_y'] - tr_d*0.1
                            min_yj = expect_position[str(j)]['min_y'] + tr_d*0.1
                        position = {'max_x':round(max_xi, 3) if abs(max_xi) < map_width/2 else map_width/2,

'min_x':round(min_xi, 3) if abs(min_xi) < map_width/2 else -map_width/2,

'max_y':round(max_yi, 3) if abs(max_yi) < map_height/2 else map_height/2,

'min_y':round(min_yi, 3) if abs(min_yi) < map_height/2 else -map_height/2}

expect_position.update({str(i):position})
                        update_flag[str(i)] = 1
                        position = {'max_x':round(max_xj, 3) if abs(max_xj) < map_width/2 else map_width/2,
```

```
'min_x':round(min_xj, 3) if abs(min_xj) < map_width/2 else -map_width/2,

'max_y':round(max_yj, 3) if abs(max_yj) < map_height/2 else map_height/2,

'min_y':round(min_yj, 3) if abs(min_yj) < map_height/2 else -map_height/2}

expect_position.update({str(j):position})
                                update_flag[str(j)] = 1
    for i in range(0, device_num):
        if not str(i) in expect_position.keys():
            position = {'max_x':map_width/2,
                        'min_x':-map_width/2,
                        'max_y':map_height/2,
                        'min_y':-map_height/2}
            expect_position.update({str(i):position})
    print(count)
    print(expect_position)
    for k, v in sorted(expect_position.items(), key=lambda x:(int(x[0]))):
        position = {}
        xd = round(((v['max_x']+v['min_x'])/2), 3)
        yd = round(((v['max_y']+v['min_y'])/2), 3)
        position = {'x':xd, 'y':yd}
        estimate.update({k:position})
estimate_distance = {}
def Estimation_Position():
    # # 定点で位置検索する手法
    # # fixed_point_x = {map_width/2, map_width*3/8, map_width/4, map_width/8, 0, -map_width/8, -map_width/4, -map_width*3/8, -map_width/2}
    # # fixed_point_y = {map_height/2, map_height*3/8, map_height/4, map_height/8, 0, -map_height/8, -map_height/4, -map_height*3/8, -map_height/2}
    fixed_point_x = {map_width*3/8, map_width/8, -map_width/8, -map_width*3/8}
    fixed_point_y = {map_height/2, map_height/4, 0, -map_height/4, -map_height/2}
    # for point_x in fixed_point_x:
    #     for point_y in fixed_point_y:
    #         for i in range(0, device_num):
    #             d = PE.calc_distance(point_x, point_y, x[i], y[i])
    #             if d < tr_d:
    #                 estimate_node = {}
    #
    #                 if str(i) not in estimate_distance:
    #                     estimate_node['calc_x'] = round(point_x, 3)
    #                     estimate_node['calc_y'] = round(point_y, 3)
    #                     estimate_node['distance'] = d
    #                 else:
    #                     estimate_node = copy.deepcopy(estimate_distance[str(i)])
    #                     estimate_node['calc_x'] = round((estimate_node['calc_x']+point_x)/2, 3)
    #                     estimate_node['calc_y'] = round((estimate_node['calc_y']+point_y)/2, 3)
    #                     estimate_node['distance'] = d
    #
    #                 # 実際の位置(確認用)
    #                     estimate_node['real_x'] = round(x[i], 3)
    #                     estimate_node['real_y'] = round(y[i], 3)
    #
    # estimate_distance.update({str(i):estimate_node})
    #
    # # 交差移動する手法
    #     route1_start = {'x':-(map_width/2), 'y':-(map_height/2)}
    # route1_goal = {'x':map_width/2, 'y':map_height/2}
    # route2_start = {'x':map_width/2, 'y':-(map_height/2)}
    # route2_goal = {'x':-(map_width/2), 'y':map_height/2}
    #
    #     transmit_node1 = {'x':route1_start['x'], 'y':route1_start['y']}
    #     transmit_node2 = {'x':route2_start['x'], 'y':route2_start['y']}
    # count = 0
    #  while transmit_node1['x']<=route1_goal['x'] and transmit_node1['y']<=route1_goal['y']:
    #      for i in range(0, device_num):
    #          d1 = PE.calc_distance(transmit_node1['x'], transmit_node1['y'], x[i], y[i])
    #          d2 = PE.calc_distance(transmit_node2['x'], transmit_node2['y'], x[i], y[i])
    #          if d1 < tr_d:
    #              estimate_node = {}
    #
    #                  # 最初に観測したときの transmit_node の座標
    #              if str(i) not in estimate_distance:
    #                  estimate_node['calc_x'] = round(transmit_node1['x'], 3)
    #                  estimate_node['calc_y'] = round(transmit_node1['y'], 3)
    #                  estimate_node['distance'] = d1
    #              # 観測中で最大、最小の座標を記録
    #              else:
    #                  estimate_node = copy.deepcopy(estimate_distance[str(i)])
    #                  if d1 < estimate_node['distance']:
    #                      estimate_node['calc_x'] = round(transmit_node1['x'], 3)
    #                      estimate_node['calc_y'] = round(transmit_node1['y'], 3)
    #                      estimate_node['distance'] = d1
    #
    #              # 実際の位置(確認用)
    #               estimate_node['real_x'] = round(x[i], 3)
    #               estimate_node['real_y'] = round(y[i], 3)
    #
    # estimate_distance.update({str(i):estimate_node})
    #
    #          if d2 < tr_d:
    #              estimate_node = {}
    #
    #                  # 最初に観測したときの transmit_node の座標
    #              if str(i) not in estimate_distance:
    #                  estimate_node['calc_x'] = round(transmit_node2['x'], 3)
    #                  estimate_node['calc_y'] = round(transmit_node2['y'], 3)
    #                  estimate_node['distance'] = d2
```

```
    #                 # 観測中で最大、最小の座標を記録
    #                 else:
    #                                         estimate_node  =
copy.deepcopy(estimate_distance[str(i)])
    #                     if d2 < estimate_node['distance']:
    #                         estimate_node['calc_x'] =
round(transmit_node2['x'], 3)
    #                         estimate_node['calc_y'] =
round(transmit_node2['y'], 3)
    #                         estimate_node['distance'] =
d2
    #
    #                 # 実際の位置(確認用)
    #                  estimate_node['real_x'] = round(x[i],
3)
    #                  estimate_node['real_y'] = round(y[i],
3)
    #
estimate_distance.update({str(i):estimate_node})
    #
    #       transmit_node1['x'] += plot_intervel[count%2]
    #                         transmit_node1['y']      +=
plot_intervel[(count+1)%2]
    #                         transmit_node2['x']      -=
plot_intervel[(count+1)%2]
    #       transmit_node2['y'] += plot_intervel[count%2]
    #       count += 1
    # 円周移動する手法
    estimate_node1 = {}
    estimate_node2 = {}
    for rad in range(360):
        for i in range(0, device_num):
            x1 = tr_d * 1.0 * math.cos(math.radians(rad))
            y1 = tr_d * 1.0 * math.sin(math.radians(rad))
            d1 = PE.calc_distance(x1, y1, x[i], y[i])
            estimate_node = {}
            if d1 <= tr_d:
                if str(i) not in estimate_node1:
                    estimate_node['calc_x'] = x1
                    estimate_node['calc_y'] = y1
                    estimate_node['distance'] = d1
                else:
                    estimate_node                    =
copy.deepcopy(estimate_node1[str(i)])
                    if d1 < estimate_node['distance']:
                        estimate_node['calc_x'] = x1
                        estimate_node['calc_y'] = y1
                        estimate_node['distance'] = d1

estimate_node1.update({str(i):estimate_node})
            x2 = tr_d * 2.0 * math.cos(math.radians(rad))
            y2 = tr_d * 2.0 * math.sin(math.radians(rad))
            d2 = PE.calc_distance(x2, y2, x[i], y[i])
            estimate_node = {}
            if d2 <= tr_d:
                if str(i) not in estimate_node2:
                    estimate_node['calc_x'] = x2
                    estimate_node['calc_y'] = y2
                    estimate_node['distance'] = d2
                else:
                    estimate_node                    =
copy.deepcopy(estimate_node2[str(i)])
                    if d2 < estimate_node['distance']:
                        estimate_node['calc_x'] = x2
                        estimate_node['calc_y'] = y2
                        estimate_node['distance'] = d2
estimate_node2.update({str(i):estimate_node})
    for i in range(0, device_num):
        estimate_node = {}
        if str(i) in estimate_node1:
            if str(i) in estimate_node2:
                if  estimate_node1[str(i)]['distance']  <
estimate_node2[str(i)]['distance']:
                    estimate_node['calc_x']              =
round(estimate_node1[str(i)]['calc_x'] + ¥

math.copysign(estimate_node1[str(i)]['distance']*1/3,
estimate_node1[str(i)]['calc_x']), 3)
                    estimate_node['calc_y']              =
round(estimate_node1[str(i)]['calc_y'] + ¥

math.copysign(estimate_node1[str(i)]['distance']*2/3,
estimate_node1[str(i)]['calc_y']), 3)
                else:
                    estimate_node['calc_x']              =
round(estimate_node2[str(i)]['calc_x'] - ¥

math.copysign(estimate_node2[str(i)]['distance']*1/3,
estimate_node2[str(i)]['calc_x']), 3)
                    estimate_node['calc_y']              =
round(estimate_node2[str(i)]['calc_y'] - ¥

math.copysign(estimate_node2[str(i)]['distance']*2/3,
estimate_node2[str(i)]['calc_y']), 3)
            else:
                estimate_node['calc_x']                  =
round(estimate_node1[str(i)]['calc_x'] - ¥

math.copysign(estimate_node1[str(i)]['distance']*1/3,
estimate_node1[str(i)]['calc_x']), 3)
                estimate_node['calc_y']                  =
round(estimate_node1[str(i)]['calc_y'] - ¥

math.copysign(estimate_node1[str(i)]['distance']*1/3,
estimate_node1[str(i)]['calc_y']), 3)
            estimate_node['real_x'] = round(x[i], 3)
            estimate_node['real_y'] = round(y[i], 3)

estimate_distance.update({str(i):estimate_node})
        elif str(i) in estimate_node2:
            estimate_node['calc_x']                      =
round(estimate_node2[str(i)]['calc_x'] + ¥

math.copysign(estimate_node2[str(i)]['distance']*2/3,
estimate_node2[str(i)]['calc_x']), 3)
            estimate_node['calc_y']                      =
round(estimate_node2[str(i)]['calc_y'] + ¥

math.copysign(estimate_node2[str(i)]['distance']*2/3,
estimate_node2[str(i)]['calc_y']), 3)
            estimate_node['real_x'] = round(x[i], 3)
            estimate_node['real_y'] = round(y[i], 3)

estimate_distance.update({str(i):estimate_node})
    print(estimate_distance)
    presume_position(estimate_distance)
PE.Neighborhood_Position()
Estimation_Position()
PE.Display_Position(estimate)
PE.Evaluation(estimate)
RS.main(estimate, signal_position, gateway_position)
```