ICS-09M-337

移動体の実時間モニタリングの為のマップマッチング方式に関する研究

指導教員　　大沢 裕教授

平成21年8月5日提出

数理電子情報系専攻　情報システム工学コース

07MM337
劉　葉

埼玉大学 理工学研究科・工学部
大沢研究室
埼玉県さいたま市桜区下大久保255

論文概要

現在，道路上を走行する車両の動きをモニタし，道路混雑状況を知ることを目的に，プローブカーに関する研究や社会実験が行なわれている．しかし，現在のプローブカーにおいては，スケーラビリティーの観点から，30 秒程度の間隔で車両の位置を捕捉することが限界となっている．しかしこの程度の間隔でのモニタリングでは，個々の車両の正確な移動経路を知ることは困難である．

そこで，各車両の動きを一定期間モニタリングし，その結果から各車両が良く通るルート情報を抽出し，その結果に基づき経路予測を立て，予測から外れたときのみ各移動体とサーバが通信を行なう方式を採用することにより，正確でスケーラビリティーの高いモニタリングが達成できると考えている．本研究は，上記の方式を実現するために必要となる，良く通るルート情報を車両の移動軌跡から抽出するための方式について研究している．

上記の目的を達成するためには，GPS を用いて取得される移動体の位置データを道路地図上に対応付けるマップマッチングが必要となる．これに関しては，1990 年代から多くの研究がなされており，それらは大きく，(1) 実時間を想定した方式と，(2) 全経路が得られた後でマッチング処理を行なう方式に大別できる．本研究での利用目的は後者のカテゴリーに属すが，従来研究では時間的に疎にサンプルされたデータを対象としており，一方，本研究では密にサンプルされたデータを用いる点が異なる．本研究では，従来方式の内 Brakatsoulas らにより提案された逐次マップマッチング法をベースに改良を加えている．更に，マップマッチングにより得られた経路を統合することにより，各車両の枝分かれを含むルート情報を記述する方式を提案し，よく通るルート情報を取得している．最終的に得られる経路は枝分かれや合流を含む経路となるが，枝分かれや合流までの単純な部分の経路を部分パスとし，その部分パスの集合をグラフとして記述する方法である．

提案方式を，長期間に亘り実際の車両の動きを計測したデータに対して適用した．GPS データは 1 秒間隔で位置を取得できる設定とし，１つのルートの平均長は約 10km である．マッチングの対象とする道路地図には，国土地理院発行の数値地図

25000 の道路データを抜き出したものを用いている．ここで使用した道路地図データは汎用地図としての利用を目的としたものであり，道路網部分には若干の接続ミスなどが存在するが，そのような地図の欠陥を除く部分においては，ほぼ 100%のマッチング率を達成している．

目次

第 1 章

序論

1.1 背景と目的

最近，プローブカーなど，車両の実時間モニタリングに関する研究が活発になっている．車両にはGPSと通信機能を備えた端末が置かれ，各車両がセンターと通信を行なうことにより，車両の位置を捕捉するものである．しかし，プローブカーでは，30秒に1回程度現在位置をセンターに伝達する方式がとられることから車両の移動経路などの正確な情報は捕捉できていない．一方，精度を向上させるためには通信頻度を上げればよいが，その場合にはスケーラビリティーの低下を引き起こす．

そこで，精度高い車両位置の捕捉をスケーラビリティー高く行なうために，車両ごとの「よく通るルート」情報を各車両とセンターが共有する枠組みに関する研究が行われている．センター側では共有するルート情報を用いて各車両の位置を予測する．一方，各車両でも同じアルゴリズムで車両位置を予測し，その予測から遅れや進みが閾値を超えたとき，車両からセンターに補正情報を伝える．また，車両が「よく通るルート」から外れたときには，センターと車両は予測アルゴリズムをdead-reckoning(推測航法) に切り替え，誤差が許容値を超えたとき，通信により同期をとる．

このような実時間モニタリング方式を実現するためには，移動体の移動軌跡を収集し，その軌跡を道路地図に対応付けるマップマッチングが重要となる．本研究では，このような用途を目的とした移動軌跡のマップマッチング方式について研究する．

マップマッチングは，GPSなどにより移動体位置が容易に計測可能になった1990年代以降，活発に研究が行われてきた[1][2][3][4]．マップマッチングの目的は利用状

況により，以下の方式に分類できる．

1. 逐次的なマップマッチング：カーナビにおいて行なわれるマップマッチングであり，例えば 1 秒毎に取得される位置データを実時間で道路上の位置に対応付ける．この場合には，多くは車載端末のような計算機パワーに劣る計算資源を用いるため，複雑な処理は行なえない．
2. バッチ的なマップマッチング：全移動軌跡データを獲得後，その移動軌跡を地図に対応付ける．この場合には，軌跡ログをもとに処理するため，複雑な処理も可能である．また，現在位置より過去の位置のほか，未来の位置を用いたマッチングを行なうことも可能である．

本論文で提案する方式は，後者のカテゴリーに属すものである．また，マップマッチングアルゴリズムは，密な位置計測の場合と疎な位置計測の場合とでも異なった方式がとられる．密な場合 (例えば 1 秒毎の計測) は，GPS の測定誤差の問題は残るものの道路セグメント上を連続的に移動する．一方，疎な計測 (例えば 30 秒毎の計測) では，隣合う 2 つの計測において，異なる道路セグメント上に対応付けられる場合があり，その間の移動軌跡を推測する必要がある．本論文で扱うマップマッチング方式は，バッチ的でありかつ密な計測によるものである．

図 1.1 にマップマッチングの原理を示す．

本研究の目的は,「よく通るルート」情報を用いた移動体の実時間モニタリングのために，蓄積された大量の移動体軌跡を道路網上に対応付け,「よく通るルート」情報を取得することにある．これを実現するために，従来提案されているマップマッチング方式の中から本目的に近い方式である Brakatsoulas 等 [13] により提案されたインクリメンタルマップマッチング方式を改良している．最後にこの結果をまとめることにより「よく通るルート」情報を抽出している．

1.2 論文構成

本論文は，7 章で構成されている．

まず，本章（第 1 章）では，本研究の背景と目的について示した．次に第 2 章において本研究に関連する研究についてまとめる．第 3〜5 章において提案方式について述べる．第 6 章では，実際の移動体による計測データを用いてマップマッチングを適用した実験結果について述べる．最後に第 7 章において本研究をまとめる．

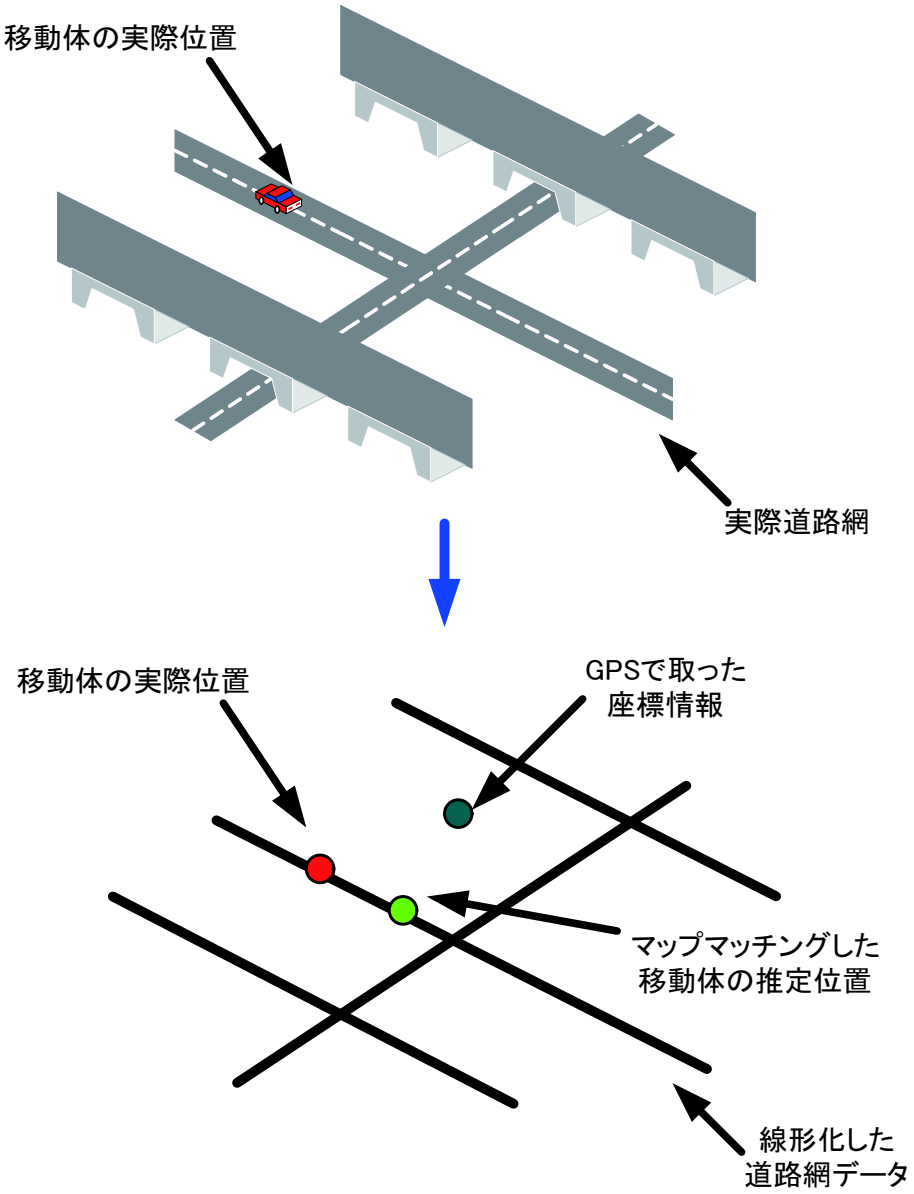


図 1.1: マップマッチングの原理図

第 2 章

関連研究

2.1 実時間モニタリング

環境問題や省資源化及などの観点から，物流や人の輸送の効率化が求められている．また，快適な生活の観点からも車移動時における情報サービスの多様化，高度化が求められている．カーナビの日本における普及率は高く、また携帯電話によるマンナビもその利便性から利用が進んでおり、バリアフリーなどへの適用も注目されている。

広くタクシーや宅配，パトカーなどの緊急車両，ロジスティックなどの分野では支配下にある車の位置をセンターで把握することが求められる．これらの応用では，経由地や目的地が予めわかっていることが多い．従って，もし車載機とセンターの双方において同じ条件で予測経路を共有し，その経路から外れた場合にのみ，その情報を伝達する（併せて双方で経路を修正する）ことにより，通信負担少なくセンター側では移動体が時々刻々と変える位置・経路を捕捉することができる．これにより先に述べたスケーラビリティー向上の問題は軽減されるものと考える．

車載器とセンターが予測経路を共有することにより，近未来の位置関係を予測する．それにより，サーバ側では新たな端末への指示や端末側に有益な POI 情報 (レストランやガソリンスタンドなどの位置情報) などをタイムリーに，または未来位置を予測して配信することができる．更に，このようにしてサーバ側で得られた移動体の経路や速度・停止位置・停止時間などの情報を時空間データベースに蓄積できる．このようにして得られた情報は，経路の分析やデータマイニングに活用できる．システムを図 2.1 に示す.

移動体のリアルタイムモニタリングに関する研究は 1990 年代から行われている。

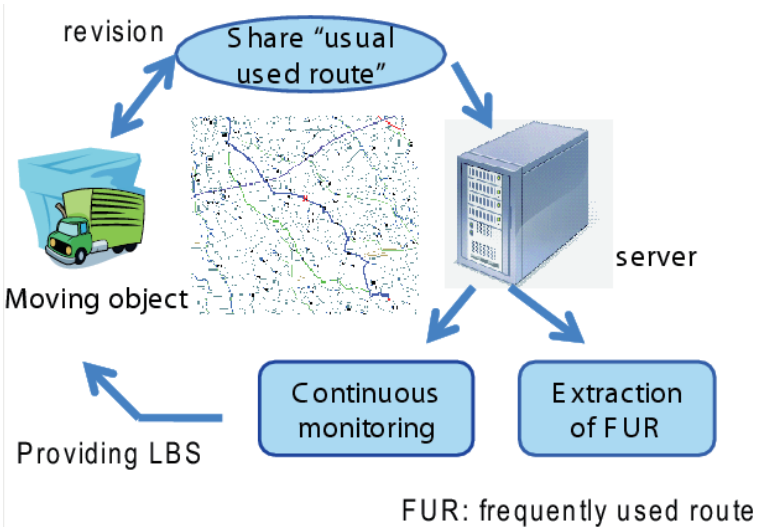


図 2.1: システムの概念図

[9][10][11]。しかし、これらは与えられたルート上を移動する移動体の位置を予測して追跡する研究であった。一般に移動体は道路ネットワーク上を自由に移動できることから、ルートに関する自由度を与えた移動体のリアルタイムモニタリング法が重要となる。

移動体が道路網上のどこを移動しているかを捕捉するためには GPS などで取得された位置を道路網上の位置に対応させる技術が必要である。これはマップマッチングと呼ばれ、従来より多数の方式が提案されてきた。[12][13][14][15][16]。道路網上での移動距離をベースとする各種空間検索の基本アルゴリズムは、Papadias 等 [17] により 2003 年に提案された。以後、道路ネットワーク上の距離に基づく k-NN 検索、範囲検索、最近接ペア、距離 Join、CkNN 等各種検索アルゴリズムが提案されている。

時間経過に伴い位置が移動するオブジェクトの管理のための時空間データ管理構造は Frentzos[18] の提案が最初と思われる。その後多くの方式が提案されている。たとえば Ding 等 [19] は道路セグメントをルートと呼ぶ単位にまとめあげ、移動軌跡の追跡を容易にする方式を提案している。

道路網上を移動するオブジェクトのリアルタイムモニタリングや、その検索に関して、最近活発に研究が行われている。Civilis 等 [20][21] は道路上での移動体の追跡法を提案している。Ku 等 [22] は移動体からの k-NN 検索法を提案している。更に、Mouratidis 等 [23] は、移動体を対象とした k-NN 検索法を提案している。また

Hsueh 等 [24] は LIT(location information table) を用いた移動体管理法を提案している。しかしこれらの方式は、移動体のルート情報などの知識を用いていない。

一方、Tiesyte 等 [25][26] は、路線に沿って移動するバスなど，ルートが限定された対象のモニタリング法を提案している。本研究では、固定したルートに限定されず、ルートからの逸脱を許している点でこれらの研究とは異なっている。

本研究では、移動体の実時間モニタリングを実行できるため、正確性及び高速化マップマッチング方式を提案する。

2.2 マップマッチングの既存方式

マップマッチングについて、大量のデータを読み込むのに対処することは、高速的なアルゴリズムを作成することが重要な目的である。ローカル空間特徴に基づく切望戦略を使用して、移動体の運行軌跡の位置サンプルが、運行軌跡のジオメトリーに従って、道路網上で選ばれた経路と比較して、順にマッチングされる。

2.2.1 基本的なアルゴリズム

移動体の運行軌跡を表している一連の位置サンプルが指定され、マップマッチングアルゴリズムは、$P2P$、$Arc2Arc$ の戦略を従って進む。

道路網上で、サンプル P_i はマッチングするため、その直前のサンプル P_{i-1} がすでにマッチングされたとすると、アルゴリズムは以下の通りに進行する。図 2.2

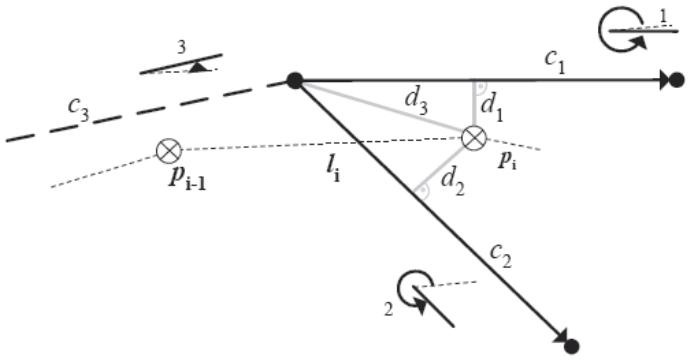


図 2.2: 基本的なマッチング戦略

最初に、P_i が対象となるサンプルに対する候補道路セグメントは C_1, C_2, C_3 であり、C_3 は直前の座標 P_{i-1} とマッチングされた。

2 つの類似性処置は、候補道路を評価するのに用いられる [27]。

S_d は、位置サンプル P_i から、各の候補道路 C_j までの距離を反映している処置であり、スケーリング要因 μ_d、n_d と距離ウェート a を用いて、下式に基づいて計算される。

$$S_d(P_i, C_j) = \mu_d - a \times d(P_i, C_j)^{n_d} \tag{2.1}$$

S_α は、候補道路に関して移動軌跡サンプルの方向を反映する処置である。それは、位置サンプル P_i に対象として、スケーリング要因 μ_a、n_a 候補道路 C_j の方向角度とサンプルベクトル P_iP_{i-1} の進行方向の差を用いて、下式に基づいて計算される

$$S_a(P_i, C_j) = \mu_a \times cos(\alpha_{i,j})^{n_\alpha} \tag{2.2}$$

それぞれ、スケーリング要因 μ_d、μ_a と n_d、n_a は S_d、S_α によって、調整される。そしたら、μ_d は μ_a より高ければ、移動体軌跡サンプルにより、距離パラメーター S_α が角度パラメーター S_d より、マッチング結果が影響させる。

本論文と実験評価より、使われる特定のデータセットのために、以下のパラメータセッティング $\mu_d = 10$ を確立したこと、$a = 0.17$、$n_d = 1.4$、そして、$\mu_a = 10$、$n_d = 4$ に設定された。すなわち、総合できな類似性定義は、各の候補道路は下式より、S_d、S_a の合計として計算される。

$$S = S_\alpha + S_d \tag{2.3}$$

これの計算値より、大きければ大きいほど、マッチングの正確性が高いである。

座標点 P_i は候補道路 C_j への予測マッチングのタイプに従い、すなわち、そのタイプは二種類が存在する。

- （ケース 1）P_i の予測マッチング位置が候補道路 C_j の真ん中にある。
- （ケース 2）P_i の予測マッチング位置が候補道路 C_j の真ん中にならない。を直すことができない。これによってマッチングの確実性に保証できない。

図 2.3 の例で、P_1 が e_1 にマッチングされた後に、アルゴリズムは P_2（ケース 1）へ進んで、また、それも e1 にマッチングされる。P_3 へ進んで、それはこの座標点を e_2 とマッチングしようとする。そして、この予測が (ケース 2) を反映する時から、それは次の位置サンプルへ進まなくて、最終的に e_3 を P_3 にマッチングする。e_2 は、横断されたエッジとして記録される。

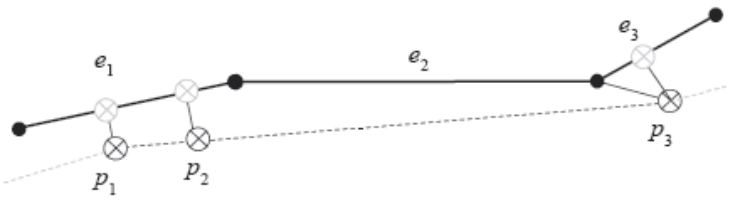


図 2.3: マッチングの例

上図で、×をつけている灰色丸は、位置サンプルがマッチングされた位置を描かれる。

マップマッチングの基本的なアルゴリズムについて手順を以下の Algorithm1 に示す。

Algorithm 1 マップマッチングの基本的なアルゴリズム

Require: サンプル P_i がマッチングした道路セグメントを取得する

1: $SPointP_i = newSPoint(ListX[i], ListY[i])$
2: $cand = stimsFiles.Index.GetTargetPolylines(P_0, dist, ent, searchLayer, diffTOI)$
3: $doubletmpMatchWetight1 = 0$
4: $doubletmpMatchWetight2 = 0$
5: $inttmpMatchNum = 0$
6: **for** $(j = 0; j < cand.Count; j++)$ **do**
7: $tmpMatchWetight1 = mapMatchingWeight(P_i, newSPoint(List[i + 1]), cand[j], false)$
8: **if** $(j == 0)$ **then**
9: $tmpMatchWetight2 = tmpMatchWetight1$
10: $tmpMatchNum = j$
11: **else if** $(tmpMatchWetight2 < tmpMatchWetight1)$ **then**
12: $tmpMatchWetight2 = tmpMatchWetight1$
13: $tmpMatchNum = j$
14: **end if**
15: **if** $(getPLStrike(cand[tmpMatchNum], P_i, newSPoint(List[i + 1])) >= 0)$ **then**
16: $exPoint = newSPoint(cand[tmpMatchNum].[cand[tmpMatchNum].ver - 1])$
17: $exPL = cand[tmpMatchNum]$
18: $matchedPL.Add(cand[tmpMatchNum])$
19: **else**
20: $exPoint = newSPoint(cand[tmpMatchNum].[0])$
21: $exPL = reversePolyline(cand[tmpMatchNum])$
22: $matchedPL.Add(reversePolyline(cand[tmpMatchNum]))$
23: **end if**
24: **end for**

1. GPS サンプルと *dist* 捜査範囲をしたがって、範囲内にある道路を $(cand)$ に挿入される。(1〜2 行目)
2. 道路の S 関数値を計算して、最大 S 関数値を持っている道路を確定する。第 8 行目、S 関数値の計算。(3〜13 行目)
3. ベクトル $P0P1$ の方向をしたがって、新たな展開点 $(exPoint)$ を確定する。P_{i+1} は距離基準を参照して映写セグメント $(exPL)$ を確定して、映写セグメ

ントはマッチング軌跡 ($matchedPL$) に挿入される。もしかしたら、マッチング軌跡の方向と GPS 軌跡の方向が異なる場合には、マッチング軌跡の方向を調整する。（15〜21 行目）

2.2.2 ローカル予測方式

上述した単純なアルゴリズムを改善するために、再帰的な「予測」方針は採用された。再帰的に、候補道路 cj ごとに、その「存在している」岐路 $C_{j,k}$ の間で、最高の S 関数値は、計算される。

ローカル予測方式は、単純な候補道路よりむしろ、代わりの候補道路につながっている岐路を調べることによってより、全体的な情勢にあっているマッチングすることを目指する。最後に、1 つの選択だけは、あっている結果で実体化される。

予測候補岐路は固定されて、本方式が、計算品質と実行コストの観点から最適であると経験的に、4 つの関係する岐路が確認された。

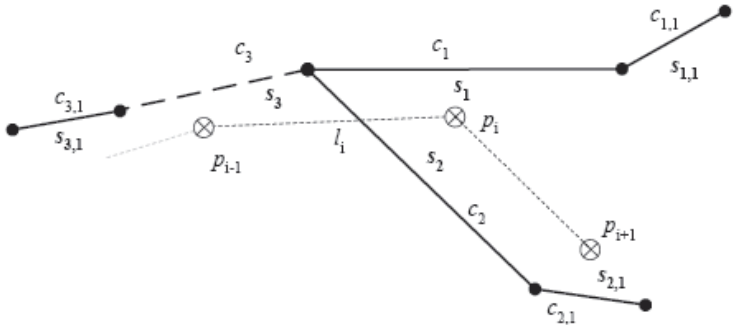


図 2.4: ローカル予測の例

図 2.4 では、2 つの関係岐路を予測するので、単純な例を表す。

サンプル P_{i-1} が C_3 にマッチングされた、対象となるサンプルが P_i になって、C_3 の岐路を展開して、C_1 と C_2 になる。P_1 を C_2 とマッチングならば、C_2 の岐路 $C_{2,1}$ は P_{i+1} にマッチするために最優の候補をわかる。すなわち、P_1 を最優の候補 C_2 とマッチングならば、P_{i+1} を C_2 の岐路（あるいは C_2 本体）にマッチするために最優の候補でもあり、P_1 を C_2 にマッチングするのは最適が確定できる。

ローカル予測方式の計算方式は下式の通りになる。

$$S(P_i, C_j) = \sum_{k,l=0}^{depth} S(P_{i+l}, C_{j,k}) \tag{2.4}$$

ローカル予測方式について手順を以下の Algorithm2, 3 に示す。

Algorithm 2 ローカル予測方式 1

Require: 各の展開した岐路の S 関数値を取得する

1: $cand2 = mmSubPath(exPoint, dist)$
2: $List < double > weightList = newList < double > ()$
3: **for** $(j = 0; j < cand2.Count; j++)$ **do**
4: **if** $(plEquals2(cand2[j][0], exPL) == false)$ **then**
5: $cand2[j].Insert(0, exPL)$
6: **end if**
7: **end for**
8: $TreeRoad = getNextPL(ListX, ListY, dist, i, step)$
9: **for** $(j = 0; j < cand2.Count; j++)$ **do**
10: $weightList.Add(0)$
11: **if** $(isAdd)$ **then**
12: $weightList[j] = weightList[j] + mapMatchingWeight(false)$
13: **else**
14: $weightList[j] = weightList[j] + mapMatchingWeight(ture)$
15: **end if**
16: **end for**
17: **for** $(j = 0; j < weightList.Count; j++)$ **do**
18: $tmpMatchWetight1 = weightList[j]$
19: **if** $(j == 0)$ **then**
20: $tmpMatchWetight2 = tmpMatchWetight1$
21: $tmpMatchNum = j$
22: **else if** $(tmpMatchWetight2 < tmpMatchWetight1)$ **then**
23: $tmpMatchWetight2 = tmpMatchWetight1$
24: $tmpMatchNum = j$
25: **end if**
26: **end for**

1. 展開点 ($exPoint$) から、展開した岐路を集合 ($cand2$) に挿入される。(1 行目)

2. 直前の映写セグメントをすべての岐路の先頭に挿入する。(2〜5 行目)
3. GPS サンプルに対して、候補道路(捜査範囲以内の道路)を集合 ($TreeRoad$) に挿入される。(8〜10 行目)
4. 各の岐路の S 関数値を計算する。(11〜24 行目)

Algorithm 3 ローカル予測方式 2

Require: ローカル予測方式でサンプル P_i がマッチングした道路セグメントを取得する

$tmpMatchNum2 = getMMPL(P_i, tmpPL)$
$polylinetmpPL2 = tmpPL[tmpMatchNum2]$
for $(l = (tmpMatchNum2 + 1); l < tmpPL.Count; l++)$ **do**
  $tmpPL.RemoveAt(l)$
  $l--$
end for
$tmpPL.RemoveAt(0)$
for $(j = (matchedPL.Count - 1); j >= 0; j--)$ **do**
  $tmpPL.Insert(0, matchedPL[j])$
end for
if $(step! = 0)$ **then**
  if $(getPLStrike(tmpPL2, P_i, newSPoint(List[i + 1])) >= 0)$ **then**
    $exPoint = newSPoint(tmpPL2.[tmpPL2.ver - 1])$
  else
    $exPoint = newSPoint(tmpPL2.[0])$
  end if
else if $(getPLStrike(tmpPL2, newSPoint(List[List.Count - 2]), p0) >= 0)$ **then**
  $exPoint = newSPoint(tmpPL2.[tmpPL2.ver - 1])$
else
  $exPoint = newSPoint(tmpPL2.[0]$
end if
$tmpPoint = newSPoint(tmpPL2.[0])$
$ppDist = get2PointDistance(oldExPoint, tmpPoint)$
if $(exPoint.equals(tmpPoint))$ **then**
  $matchedPL = tmpPL$
  $exPL = tmpPL2$
  $break$
else
  $matchedPL = tmpPL$
  $exPL = tmpPL2$
  $break$
end if

1. 最優先岐路を確定する。(1 行目)
2. 対象としてサンプルの映写セグメントを確定する。(2 行目)
3. 映写セグメントと最優先岐路の組み合わせ(結果は総合軌跡)。(3〜5 行目)
4. 総合軌跡と直前のマッチング軌跡の組み合わせ(結果は更新したマッチング軌跡)。(7〜9 行目)
5. 対象として展開点の確定。(11〜19 行目)
6. 映写セグメントの終点を確定する。(20〜21 行目)
7. 展開点とマッチング終点の比較を行う。展開点とマッチング終点が同一の場合には、マッチング成功した、マッチング軌跡を戻す。異なる場合には、対象として岐路の候補を消して、最優先岐路をもう一回を確定する。(22〜30 行目)

2.2.3 位置サンプルの初期化

上記のあっている戦略を適用するために、アルゴリズムは最初の位置サンプル P_0 をデータ地図にマッチングすることによって、初期化される必要がある。

アルゴリズムを使用することができるために、ベクトル P_0P_1 の進行方向を確定して、マッチングする候補道路が集合 E_0 に集めされる。

単純なアプローチを使って、集合 E_0 はサンプル P_0 探索範囲内であるすべての道路セグメントを含む。

探索範囲は GPS エラーに関係があって、本研究の実験データが適用できるように $30M$ に確定された。E_0 に含むすべての道路セグメントは、それから、P_0 にマッチングするものを決定するために、S 関数値を計算して評価される。

2.2.4 パフォーマンスの評価

N 個のマッチングサンプルを置いて、アルゴリズムは各々のサンプルのために評価して、a 本の候補道路につながっている有限数の岐路 l 本に従って、その実行コストは、$O(na^{l+1})$ である。

A, l 両方とも定数として、それを与えれば、アルゴリズムは効果的に $O(n)$ 時間で動く。

初期化コストは、位置に最も近い道路セグメントの集合を見つけることによって決定される。隣接関係リストを使って、この検索は $O(logv + w)$ において成し遂げられることができる。そこで、v は道路ネットワークで頂点の個数で、そして、確

定した集合のサイズは w である。

つまり、実際のマップマッチングコスト $O(n)$ であり、初期化のコストは $O(logv + w)$ になる。

2.3 移動体軌跡データ

移動体データは動体軌跡に関してモデル化されることができるし、そして、それは位置サンプルを差し込むことによって得られる。一般的に、一次補間が多項式のスプラインのような他の方法と対照的に使われます。

位置サンプルはそれから多角形の線分のエンドポイントになって、そして、すべての移動体の動きは多角形によって $3D$ 空間で見受けられる [8]。

位相対の位置決め技術は、一般的に GPS であり、サンプリングレートと関連したその合同測定エラーは、位置サンプルを道路網とマッチングことを要求する。

2.3.1 測定エラー

通常に 2 つの評価は、GPS の正確さについてなされる。

1. エラー配布（すなわち 3 次元の各々におけるエラーと時間内のエラー）は、ガウスであるとされる。
2. 水平エラー配布（すなわち $X - Y$ 空間の配布）が円を描くと仮定することができる [28]。

位置 GPS 測定におけるエラーは、二変量の正規分布される後で、確率関数によって説明されることができて、確率関数は、それぞれの 2 次元空間で、2 つの正規分布から成る。

図 2.5 では、エラー配布を視覚化して、GPS 精度（例えば正確さが言及する $5M$ の GPS）に言及するとき、平均標準偏差に加えて、述べられる特徴のあるパラメーターはある。平均には、$\pm D$ の範囲で、二変量の正規分布において、39.35% の可能性は集中される。

GPS エラーが特定の状況（陰になって反射された信号）で相当でありえるが、GPS 信号（WAAS、EGNOS）を増やすことを使って、典型的エラーは $8M - 2M$ の範囲にある。

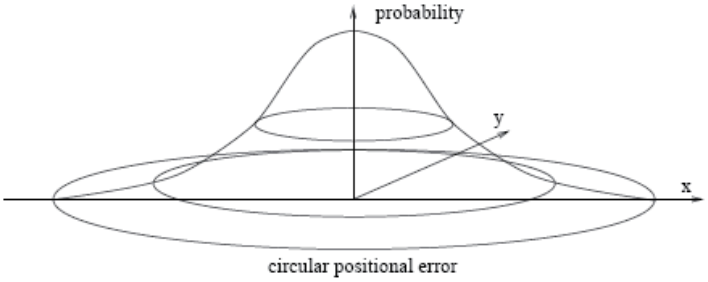


図 2.5: エラー配布

2.3.2 移動体軌跡データの管理

移動体軌跡データの正確さを除いて、マップマッチング結果の最終的なアプリケーションは、それぞれのアルゴリズムのデザインに影響を及ぼす。交通評価と予測のためのデータソースとして移動体軌跡データを利用する際に、実際地図はグラフとしてモデル化されて、辺と頂点は道路網を構成する。

2.4 マップマッチング既存方式の欠点

既存方式で、マップマッチングする前に、GPS サンプルの処理が行っていない。

1. GPS サンプルが密集している場合にマッチングミスが多く、実行時間も長い
2. サンプルに大きな誤差があるとき、マッチングミスが多い

既存方式で、初期道路ポリラインのマッチングミスが多い。既存方式の初期道路ポリラインのマッチング方式は、単純的に、式 $S = S_\alpha + S_d$ を用いて、確定すればマッチングミスが多いである。

既存方式で、GPS 軌跡と道路セグメントは交差および道路交差点で、マッチングミスが多い。既存方式は、GPS 軌跡と道路セグメントは交差および道路交差点でマッチングに対応する処理がないので、交差と交差点の場合でマッチングミスが多いである。

第 3 章

提案方式（GPS サンプルの処理）

道路セグメントおよび地形は複雑な区域上で、高層ビル、陸橋、あるいはトンネルたくさんある原因で、GPS の受信機が一部の人工衛星から発信した情報を受信することを失わせる。取られた移動体の位置情報はドリフトを生じて、これによって移動体の位置決めをして誤差か出てきてしまった。

GPS の定位実験結果より、取られた移動体の位置情報はある時刻にすごく大きな変動が発生する可能である。これらのすごく大きな変動を引き起こした取られた移動体の位置情報は「ノイズ」と呼ばれる。

この異常データがナビゲーションおよびマップマッチングに不良影響を避けることとして、マップマッチングして前は、有効な方法を用いて、それらのノイズを検出して取り消しなければいけないである。

ノイズの取り消しに関する要素が以下の通りになる。

1. 利用できる GPS 人工衛星の数量
2. HDOP の精度値
3. 地図データのトポロジー相関性と移動体の運行速度

一般的には、ノイズの取り消しが組み合わせる伝定位方式を採用される。たとえば、GPS/DR が組み合わせる定位方式、ナビゲーション推定修正方式、GPS 定位を用いて DR 誤差を直す方式などである。

ただ本論文によった、ターミナルを移動体に基づいて、経済性と便利性から言うことにも拘らず、ハードウェア方式を採用することはノイズの削除が実行不能である。応用の便利性および持ち合わせの GPS 精度に配慮して、アルゴリズムを用いて GPS データ処理を採用する。

3.1　停止時のサンプル点の間引き

移動体の時速は、マップマッチングに対して、非常に影響になっている。移動体の進行時速は、直接に GPS サンプルの品質に関係ある。具体的に言えば、移動体時速は非常に低いならば（交通信号、交差点、進行途中の停止)、GPS サンプルが密集しており、データ品質はよく悪いである。この原因でマッチングを行っているとき、ミスあるいは無駄の処理、岐路の展開が起こってしまう。

マッチング正確性の保証及び実行コストの高速化のために、移動体時速範囲の拘束を提案する。

3.1.1　移動体時速範囲の設定

本研究と実験評価より、使われるさいたま市 25,000 データセットのために、以下の移動体時速範囲を設定した。図 3.1

図 3.1: 移動体時速範囲の設定

移動体時速範囲の拘束について手順を以下の Algorithm4 に示す。

Algorithm 4 移動体時速範囲の拘束

Require: 時速は $(0.4km/h, 80km/h)$ 範囲以内の GPS サンプルが削除される

```
1: for (i = 0; i < TraListX[idx].Count − 1; i++) do
2:     V = getVelocity(i, i + 1, idx)
3:     Consile.Writeline(V + "km/h")
4:     if (V < minV||V > maxV) then
5:         TraListX[idx].RemoveAt(i + 1)
6:         TraListY[idx].RemoveAt(i + 1)
7:         ListT[idx].RemoveAt(i + 1)
8:         i − −
9:     end if
10: end for
```

1. すべての GPS サンプルに対して、指定された対象としてサンプルとこの直後のサンプルの時間と距離間隔を参照して、移動体の時速を取得する。(1〜3 行目)
2. 取得した移動体時速と指定された時速範囲を比較する。(4 行目)
3. 指定された時速範囲にあっていないサンプルを削除する。(5〜8 行目)

3.1.2 処理結果

処理結果より、処理した GPS サンプル軌跡と元の軌跡はまったく一緒である。また、サンプルが密集しているところには、無効のサンプルが多くとも 70% を取り消した。

つまり、結果よりマッチング正確性を影響せずに、マッチング実行が高速化できたといえる。図 3.2

3.2 隣接する二点のサンプル距離を用いた間引き

移動体は実際進行軌跡がさまざまである。その中、移動体は迂回の状況がよく発生している。迂回軌跡に対して、GPS サンプルの異常ではなくて、移動体の時速異常も言えないである。

移動体迂回は、一般的には、移動体が間違った経路を自体に修正する。あるいは、移動体が向きを 180 度かえるとき、発生する。

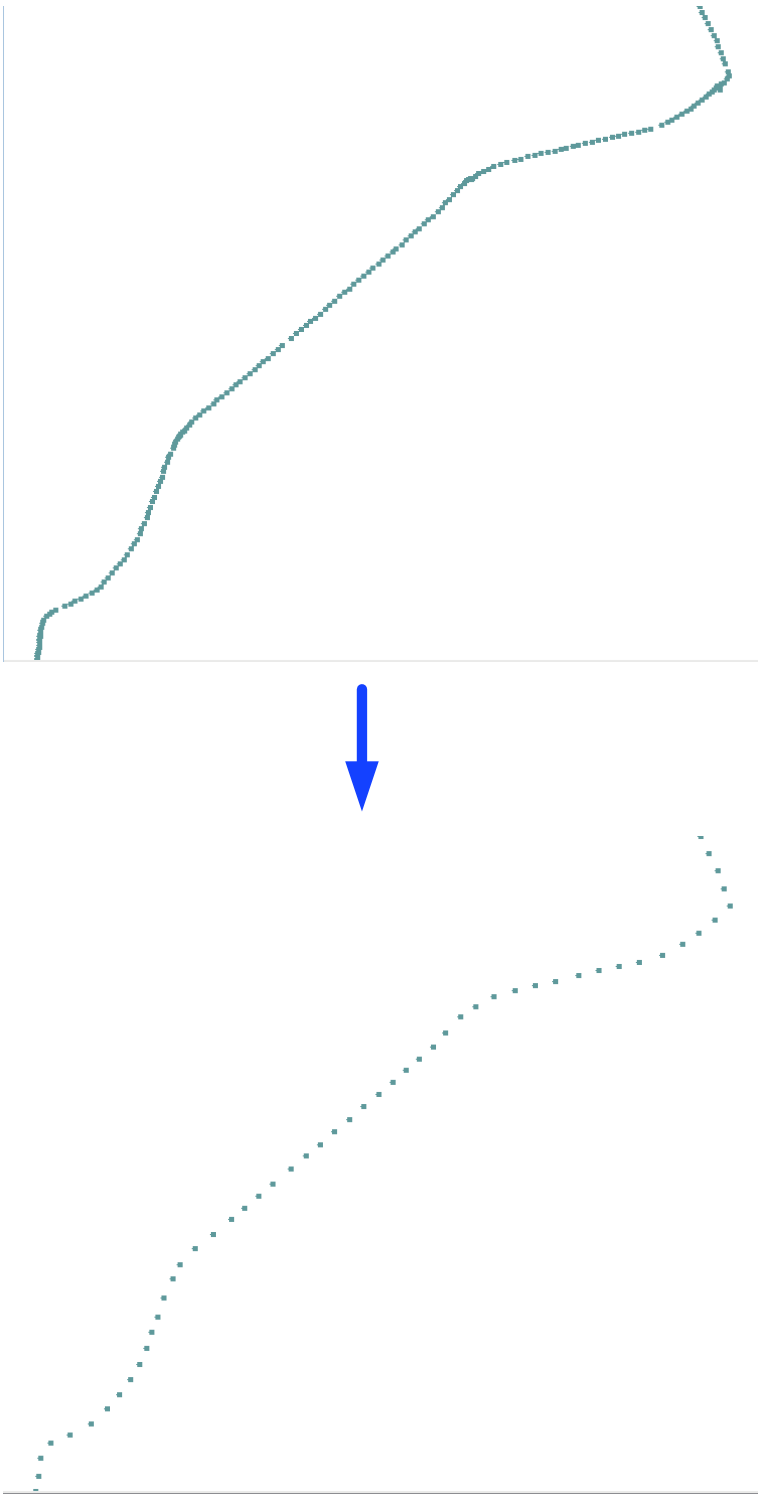


図 3.2: 処理結果

上記のサンプルも直接にマッチングの品質に影響して、実行コストを低速化の原因の一つである。

そしたら、マッチング正確性の保証及び実行コストの高速化のために、隣接の二点サンプルの距離間隔の拘束を提案する。

3.2.1 隣接の二点サンプルの距離間隔の設定

本研究と実験評価より、使われるさいたま市 25,000 データセットのために、以下の隣接の二点サンプルの距離間隔を設定した。図 3.3

図 3.3: 隣接の二点サンプルの距離間隔の設定

隣接の二点サンプルの距離間隔の拘束について手順を以下の Algorithm5 に示す。

Algorithm 5 隣接の二点サンプルの距離間隔の拘束

Require: 距離間隔は $25m$ 範囲以内の GPS サンプルが削除される

1: **for** $(i = 0; i < TraListX[idx].Count - 1; i++)$ **do**
2: $V = getDist(i, i+1, idx)$
3: $Consile.Writeline(V + "m")$
4: **if** $(V < minD)$ **then**
5: $TraListX[idx].RemoveAt(i+1)$
6: $TraListY[idx].RemoveAt(i+1)$
7: $ListT[idx].RemoveAt(i+1)$
8: $i--$
9: **end if**
10: **end for**

1. すべての GPS サンプルに対して、指定された対象としてサンプルとこの直後のサンプルの距離間隔を取得する。(1〜3 行目)

2. 取得した距離間隔と指定された距離間隔範囲を比較する。(4 行目)
3. 指定された距離間隔範囲にあっていないサンプルを削除する。(5〜8 行目)

3.2.2 処理結果

処理結果より、処理した GPS サンプル軌跡と元の軌跡はまったく一緒である。また、サンプルが密集しているところまたは、迂回原因で GPS 軌跡上の小さい分岐を修正できた。無効のサンプルが多くとも 78% を取り消した。

つまり、結果よりマッチング正確性を影響せずに、マッチング実行が高速化できたといえる。図 3.4

3.3 隣接する三点のサンプルが成す角を用いた間引き

上述した、GPS サンプルの異常はいくつかの要素に関係ある。サンプル異常の表現は、サンプル軌跡の変動、サンプルの飛び出しである。サンプル異常が発生したら、マッチング失敗の可能性が急劇に増える。

そしたら、マッチング正確性の保証及び実行コストの高速化のために、隣接の三点サンプルのなす角の拘束を提案する。

3.3.1 隣接の三点サンプルのなす角の設定

本研究と実験評価より、使われるさいたま市 25,000 データセットのために、隣接の三点サンプルのなす角を設定した。図 3.5

隣接の三点サンプルのなす角の拘束について手順を以下の Algorithm6 に示す。

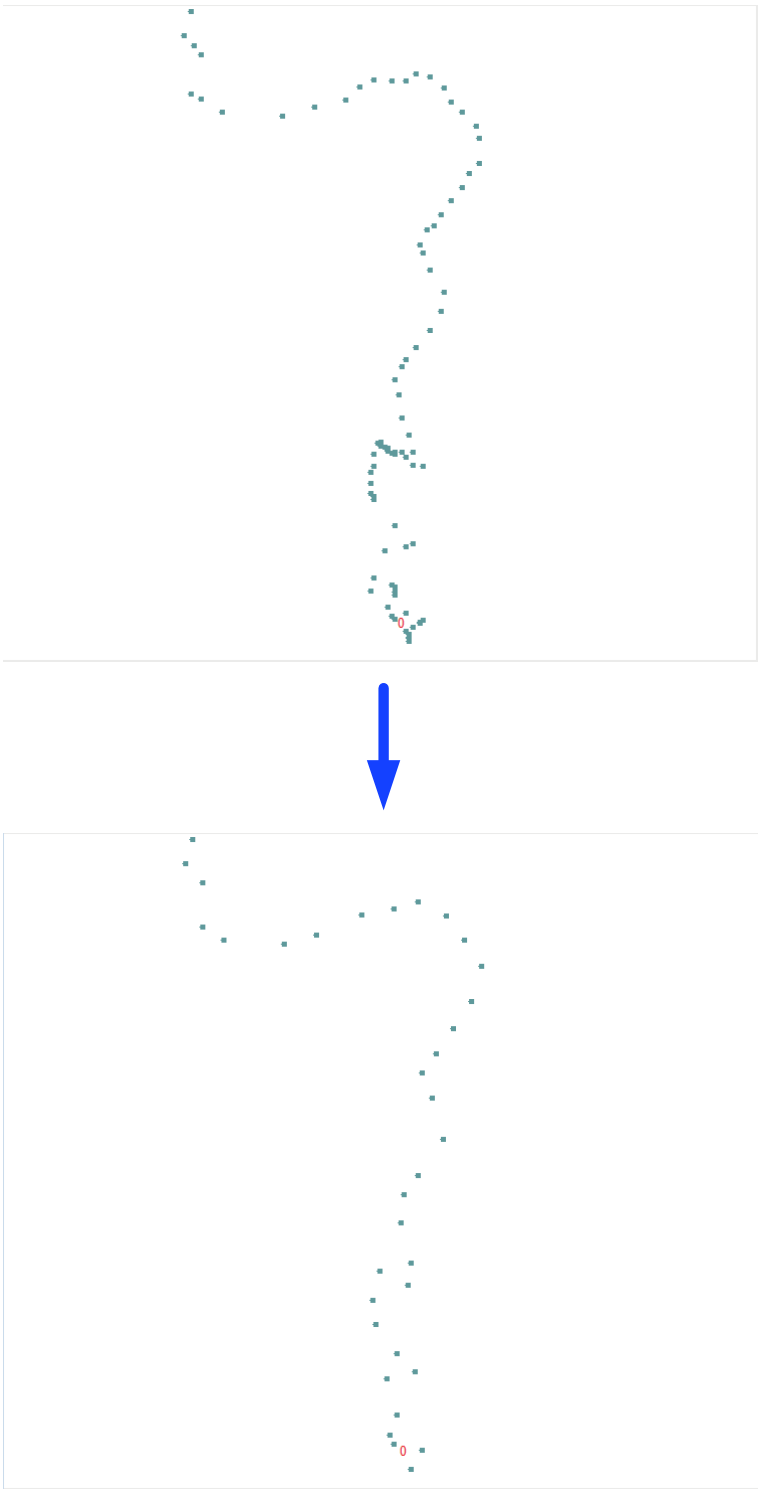


図 3.4: 処理結果

図 3.5: 隣接の三点サンプルのなす角の設定

Algorithm 6 隣接の三点サンプルのなす角の拘束

Require: なす角は $0.79rad$ 範囲以内の GPS サンプルが削除される

1: **for** $(i = 0; i < TraListX[idx].Count - 2; i++)$ **do**
2: $doubleR = getRad(i, i+1, i+2, idx)$
3: **if** $(R < minR)$ **then**
4: $TraListX[idx].RemoveAt(i+1)$
5: $TraListY[idx].RemoveAt(i+1)$
6: $ListT[idx].RemoveAt(i+1)$
7: $i--$
8: **end if**
9: **end for**

1. すべての GPS サンプルに対して、指定された対象としてサンプルとこの直後の 2 個のサンプルが構造したなす角を取得する。(1〜2 行目)
2. 取得したなす角と指定されたなす角範囲を比較する。(3 行目)
3. 指定されたなす角範囲にあっていないサンプルを削除する。(4〜7 行目)

3.3.2 処理結果

処理結果より、はっきりと見て取れる異常を修正できた。異常の取り除くのが詳しい道路状況及びサンプル品質によって、なす角を設定しなければならないである。図 3.6

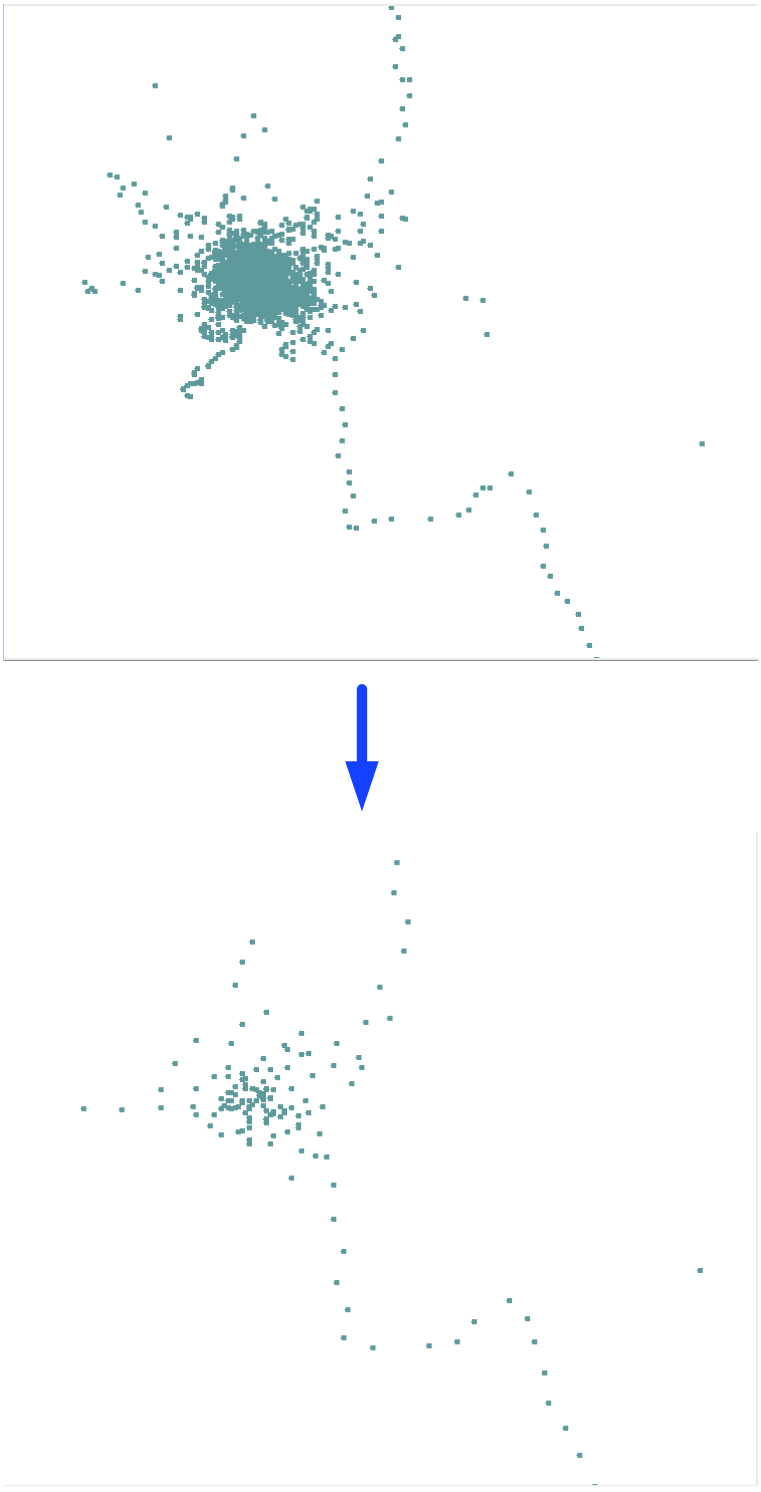


図 3.6: 処理結果

第 4 章

提案方式（初期マッチング方式の改良）

既存方式の初期道路ポリラインのマッチング方式は、単純的に、式 $S = S_\alpha + S_d$ を用いて、確定すればマッチングミスが高いである欠点を考えして、本章では、既存方式で初期ポリランマッチングミス検討の基礎に、改良した方式を提案する。提案方式で取った実験結果より、初期道路ポリラインのマッチング正確性を高めて、全体的な正しいマッチング軌跡を取得するのを保証できる。

4.1 既存方式で初期ポリラインマッチングミスの表現

始点（P_0）の進行角度は、ベクトル P_0P_1 の進行方向が確定する。移動体進行の初期は時速、取ったサンプルの時間、距離間隔がすべて理想ではないで、GPS サンプル処理は行ったとしても、やはり初期ポリラインをマッチングミスした。図 4.1

4.2 初期道路ポリラインのマッチング方式の改良

上記の原因を考えして、初期道路ポリラインのマッチング正確性を高めるために、改良方式を提案する。提案方式とは、ベクトル $P_0P_1, P_1P_2, P_2P_3, P_3P_4$ の平均角度が、始点 P_0 の方向角度に設定される。

実験結果より、初期道路ポリラインのマッチング正確性を高めできた。図 4.24.2

初期道路ポリラインのマッチング方式手順を以下の Algorithm7 に示す。

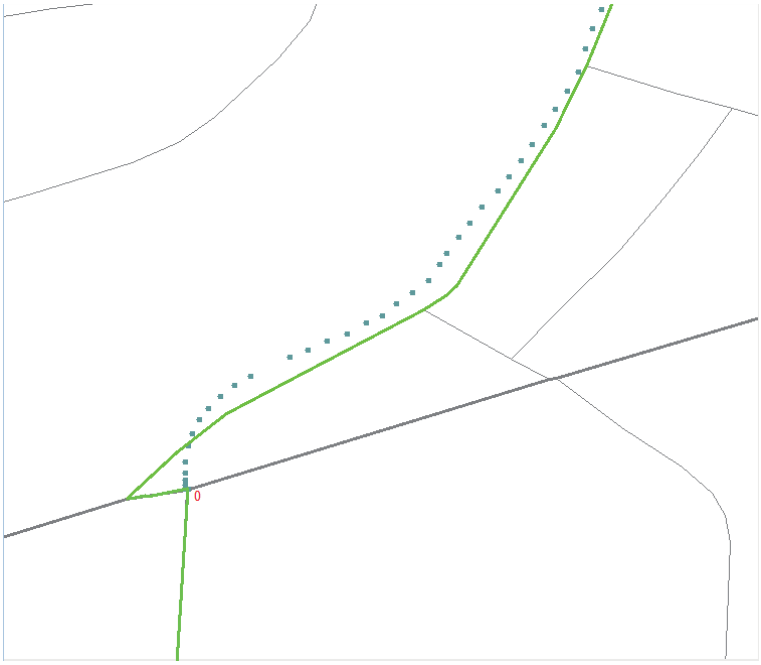


図 4.1: 初期ポリラインマッチングミス

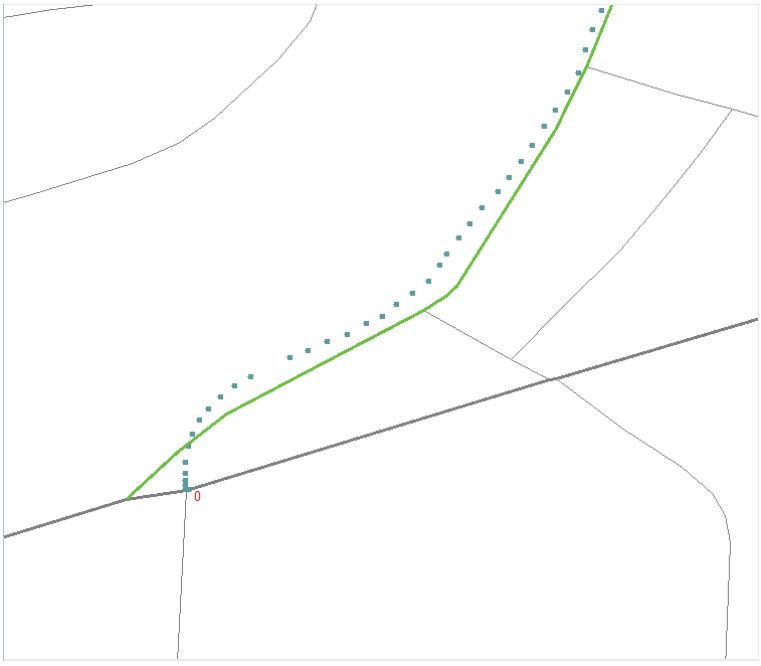


図 4.2: 初期ポリラインマッチング成功例

Algorithm 7 初期道路ポリラインのマッチング方式

Require: 始点と始点の直後 3 点の平均進行角度を取得する

1: $doubleavAngle = 0$
2: $intindexPNum = 4$
3: **if** $(ListX.Count < 4)$ **then**
4: **for** $(j = 0; j < (indexPNum - ListX.Count); j++)$ **do**
5: $indexPNum--$
6: **end for**
7: **end if**
8: **for** $(j = 0; j < indexPNum; j++)$ **do**
9: **if** $(j == 0)$ **then**
10: $avAngle = getSeparationAngle(newSPoint(List[j]), newSPoint(List[j+1])) * 2$
11: **else**
12: $avAngle = avAngle + getSeparationAngle(newSPoint(List[j]), newSPoint(List[j+1]))$
13: **end if**
14: **end for**
15: $avAngle = avAngle/indexPNum$

1. GPS サンプルの平均進行角度を設定する。(1 行目)
2. 参照する GPS サンプルの個数を確定する。(2 行目)
3. GPS サンプルは確定された参照個数が足りない場合には、すべての存在しているサンプルを参照する。(3〜5 行目)
4. 参照されたサンプルの平均進行角度を計算する。(8〜15 行目)

第 5 章

提案方式（よく通る経由頻度及び移動平均時速の抽出）

精度高い車両位置の捕捉をスケーラビリティー高く行なうために、車両ごとの「よく通るルート」情報を各車両とセンターが共有する枠組みを提案する。センター側では共有するルート情報を用いて各車両の位置を予測する。一方、各車両でも同じアルゴリズムで車両位置を予測し、その予測から遅れや進みが閾値を超えたとき、車両からセンターに補正情報を伝える。また、車両が「よく通るルート」から外れたときには、センターと車両は予測アルゴリズムを dead-reckoning(推測航法) に切り替え、誤差が許容値を超えたとき、通信により位置の同期をとる。

上記の提案を実現するため、本研究では、自宅や勤務先を起点として車で移動するとき、日常的に良く訪れる複数の目的地が存在する。また同じ目的地に至るルートも複数存在し得る。ある車の移動経路を一定期間観測することにより、よく通るルート情報が得られる。その経路の部分に対して、平均的な移動速度を付与する。この情報をサーバと移動体で共有する。

5.1 よく通るルート経由頻度の抽出

マップマッチングによって得られた経路データよりデータウェアハウスを作成した。作成したデータウェアハウスを地図上で表示したものである。よく通るルート上で、分岐点を表し、座標が表示される。各の分岐路に対して、通過回数の多さに合わせてポリラインの太さを太く表示している。図 5.1

よく通るルートの表示方式の手順を以下の Algorithm 8 に示す。

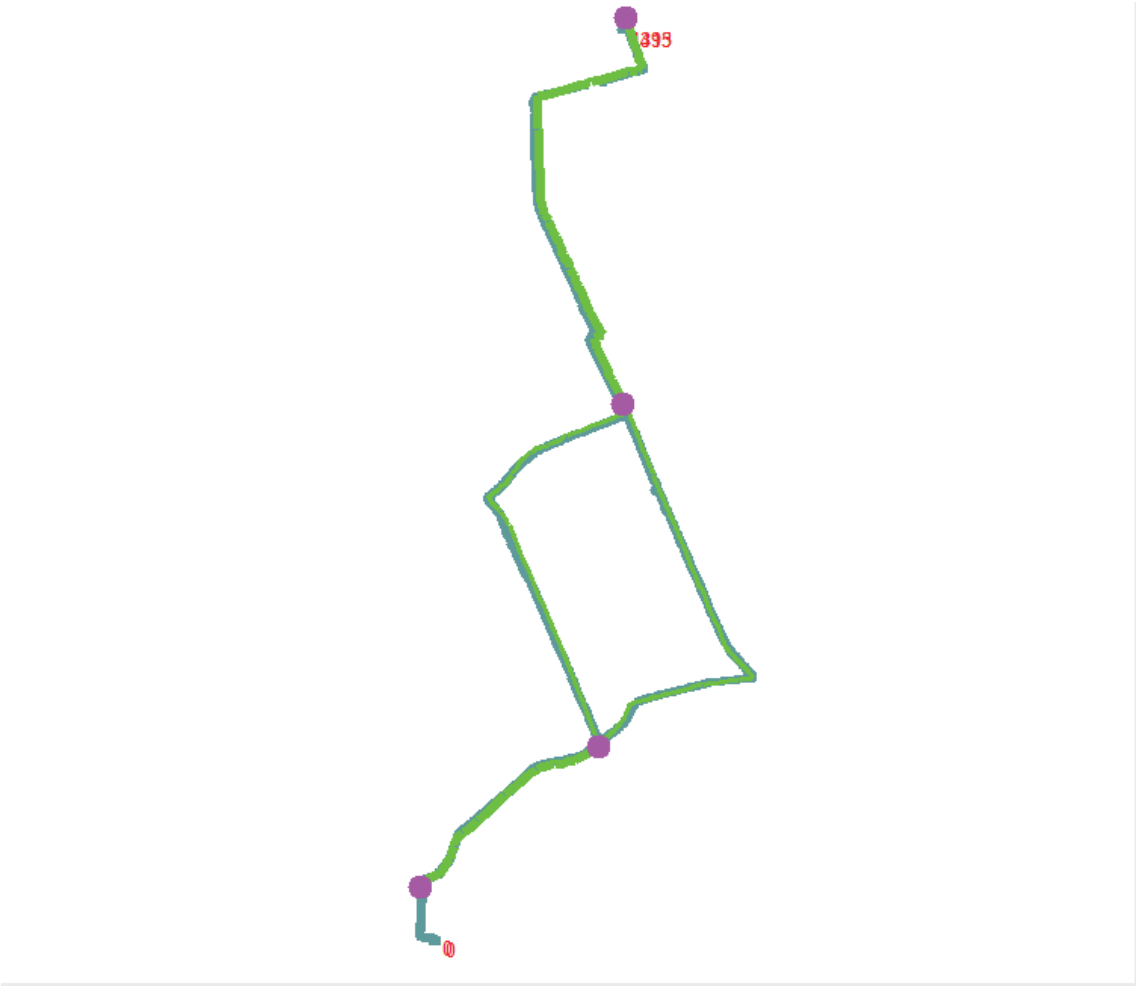


図 5.1: よく通るルートの表示

Algorithm 8 よく通るルートの表示

Require: マップマッチングしたよく通るルートの表示

```
1: privatebooldrawAll = false
2: privateList<TreePoint> muchPLs = newList<TreePoint>()
3: privateList<SPoint> mplSPs = newList<SPoint>()
4: privateList<SPoint> mplEPs = newList<SPoint>()
5: privateList<SPoint> mmps = newList<SPoint>()
6: privatevoidMNAndF()
7: for (i = 0; i < matchedPath.Count; i++) do
8:    for (j = 0; j < matchedPath[i].Count; j++) do
9:       for (k = (j + 1); k < matchedPath[i].Count; k++) do
10:         if (plEquals2(matchedPath[i][j], matchedPath[i][k])) then
11:            matchedPath[i].RemoveAt(k)
12:            j − −
13:            break
14:         end if
15:      end for
16:   end for
17: end for
18: mplEPs.Clear()
19: mplSPs.Clear()
20: mmps.Clear()
21: mmPoints.Clear()
22: for (i = 0; i < matchedPath.Count; i++) do
23:   mplSPs.Add(null)
24:   mplEPs.Add(null)
25:   if (matchedPath[i].Count > 0) then
26:      if (matchedPath[i].Count == 1) then
27:         mplSPs[i] = newSPoint(matchedPath[i][0].x[0], matchedPath[i][0].y[0]
28:         mplEPs[i] = newSPoint(matchedPath[i][0].x[matchedPath[i][0].ver −
            1], matchedPath[i][0].y[matchedPath[i][0].ver − 1])
29:         mmps = insertPoint(mmps, mplSPs[i])
30:      else
31:         mplSPs[i] = newSPoint(matchedPath[i][0].x[0], matchedPath[i][0].y[0])
32:         mmps = insertPoint(mmps, mplSPs[i])
33:      end if
34:   end if
35: end for
```

1. よく通るルートマイニングの設定（1 行目）
2. 分岐点の始点と終点集合及び重複数量の設定（2 行目）
3. 道路セグメント始点集合の作成（3 行目）
4. 道路セグメント終点集合の作成（4 行目）
5. 一時的な分岐点集合の作成（5 行目）
6. マイニングするようの座標の設定（6 行目）
7. マッチング軌跡上で重複した道路セグメントの削除（7〜17 行目）
8. 各の集合をクリアする（18〜21 行目）
9. 各の道路セグメントの始点と終点の座標を取得し、経由道路を確定する（19〜35 行目）

二つの隣接している分岐点の間には、存在している分岐路について、移動体が各の分岐路上で始点座標、終点座標及び経由回数が計算され表示している。

抽出した通過回数に関する情報が CSV 形式で出力される。図 5.2

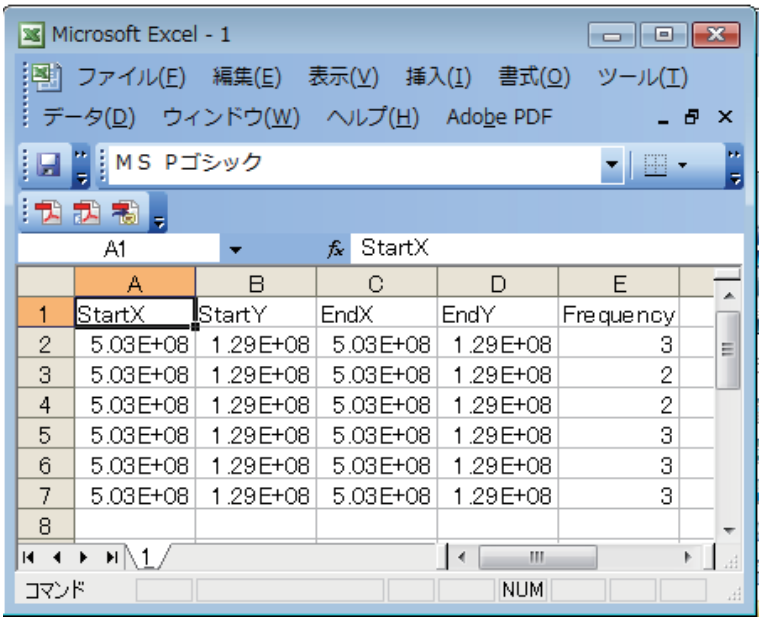


図 5.2: 通過回数の CSV

よく通るルート経由頻度の抽出手順を以下の Algorithm 9 に示す。

Algorithm 9 よく通るルート経由頻度の抽出

Require: よく通るルート経由頻度の抽出

```
1: for (j = 0; j < matchedPath.Count; j++) do
2:   boolinMuch = false
3:   if (muchPLs[i].Point1.equals(matchedPath[j][k].x[0], matchedPath[j][k].y[0])||muchPLs[i].P
     1], matchedPath[j][k].y[matchedPath[j][k].ver − 1])) then
4:     inMuch = true
5:     if (k < (matchedPath[j].Count − 1)) then
6:       if (muchPLs[i].Point1.equals(matchedPath[j][k + 1].x[0], matchedPath[j][k+1].y[0])||muchPLs[i].Point1.equals(matchedPath[j][k+1].x[matchedPath[j][k + 1].ver − 1], matchedPath[j][k + 1].y[matchedPath[j][k + 1].ver − 1])) then
7:         mp1 = k + 1
8:       else
9:         mp1 = k
10:      end if
11:    else
12:      mp1 = k
13:    end if
14:  end if
15: end for
16: mmPoints.Clear()
17: for (i = 0; i < muchPLs.Count; i++) do
18:   SPointtmpPoint = muchPLs[i].Point1
19:   SPointtmpPoint = muchPLs[i].Point2
20:   stringdrawStr = ""
21:   stringdrawStrf = ""
22:   stringnumStrf = ""
23: end for
```

1. 経由回数を取得する（1〜15 行目）
2. 経由した道路に対して、始点と終点座標及び経由回数を描く（8〜15 行目）

5.2 移動平均時速の抽出

よく通るルートに存在しない道路に逸れたときは、道路に沿って一定速度で移動する予測 (dead-reckoning) に切り替えるのを実現できるように、移動平均時速の抽出を行う。

5.2.1 移動平均時速の計算手法

各の道路セグメントに対して、移動平均時速の計算するため、移動時間を取得必要がある。

移動時間を取得する手法について、道路セグメントの端点に対して、最も近くに存在している GPS サンプルを確定して、このサンプルの取った時間 (T_{P_i}, T_{P_j}) も取得する。

道路セグメントに対して、移動時間が下式をしたがって計算される。

$$T_{C_i} = T_{P_j} - T_{P_i} \tag{5.1}$$

移動平均時速が下式をしたがって計算される。

$$AverageV_{C_i} = \frac{dist_{C_i}}{T_{C_i}} \tag{5.2}$$

移動時間を取得するため、参照 GPS サンプルを確定する。図 5.3

5.2.2 移動平均時速を抽出した表示結果

よく通るルート上で、一つ一つの道路セグメントに対して、セグメント ID、経由の始点終点、経由時間、及び通過平均時速が表示された。図 5.4

抽出した通過平均時速に関する情報が CSV 形式で出力される。図 5.5

移動平均時速を抽出する手順を以下の Algorithm 10 に示す。

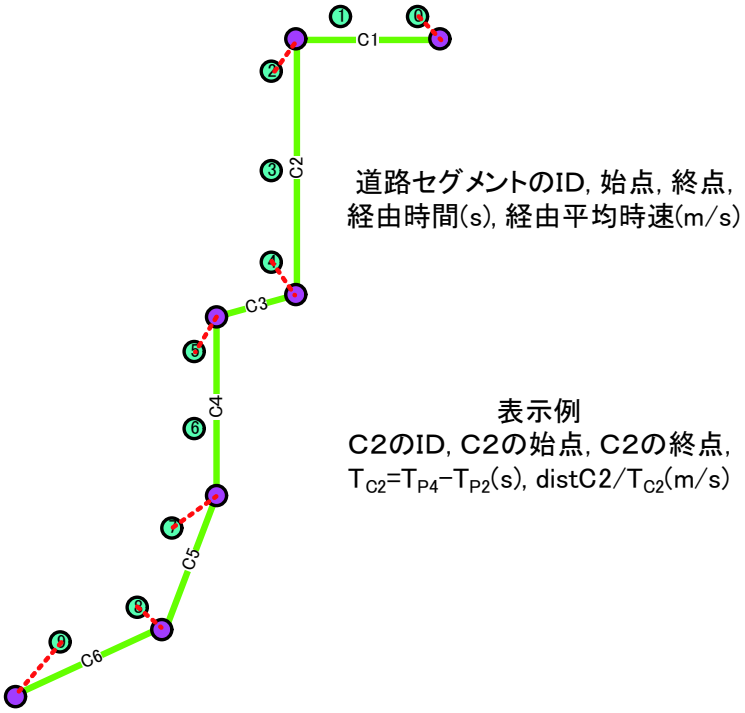
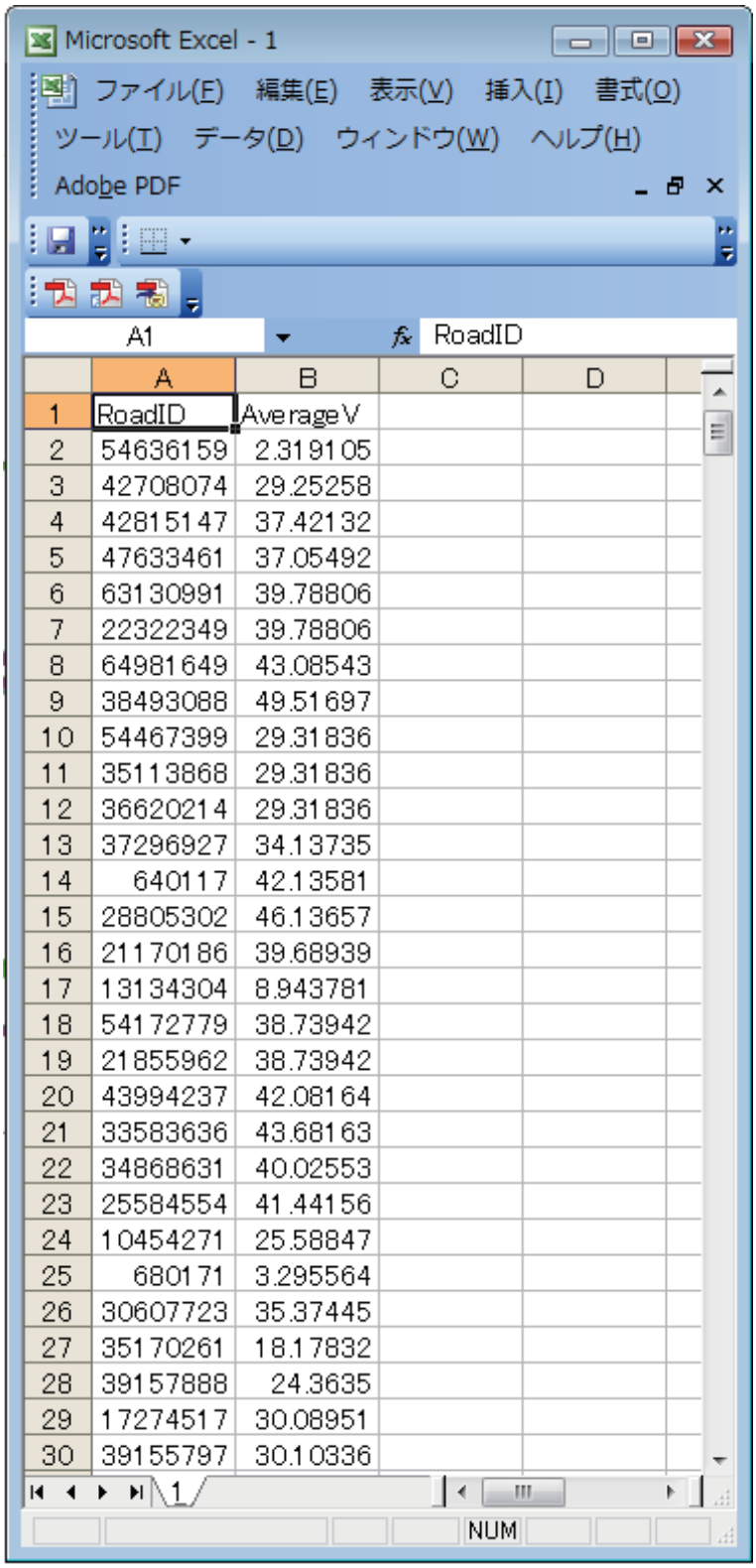


図 5.3: 参照 GPS サンプルの確定

図 5.4: 移動平均時速の抽出



	A	B	C	D
1	RoadID	AverageV		
2	54636159	2.319105		
3	42708074	29.25258		
4	42815147	37.42132		
5	47633461	37.05492		
6	63130991	39.78806		
7	22322349	39.78806		
8	64981649	43.08543		
9	38493088	49.51697		
10	54467399	29.31836		
11	35113868	29.31836		
12	36620214	29.31836		
13	37296927	34.13735		
14	640117	42.13581		
15	28805302	46.13657		
16	21170186	39.68939		
17	13134304	8.943781		
18	54172779	38.73942		
19	21855962	38.73942		
20	43994237	42.08164		
21	33583636	43.68163		
22	34868631	40.02553		
23	25584554	41.44156		
24	10454271	25.58847		
25	680171	3.295564		
26	30607723	35.37445		
27	35170261	18.17832		
28	39157888	24.3635		
29	17274517	30.08951		
30	39155797	30.10336		

図 5.5: 通過平均時速の CSV

Algorithm 10 移動平均時速の抽出

Require: 各の道路セグメントに対して移動平均時速の抽出

```
for (i = 0; i < matchedPath.Count; i++) do
    if (matchedPath[i].Count > 0) then
        List<SPoint> tmpPList = newList<SPoint>()
        List<int> tmpVList = newList<int>()
        List<SPoint> tmpPList = newList<SPoint>()
        SPointtmpPoint = null
        SPointp1 = null
        SPointp2 = null
    end if
end for
if (matchedPath[i].Count > 1) then
    for (j = 0; j < matchedPath[i].Count; j++) do
        if (j == 0) then
            p1 = newSPoint(matchedPath[i][j].x[0], matchedPath[i][j].y[0])
            p2 = newSPoint(matchedPath[i][j].x[matchedPath[i][j].ver − 1], matchedPath[i][j].y[matchedPath[i][j].ver − 1])
        else
            if (j == 1) then
                if (p1.equals(matchedPath[i][j].x[0], matchedPath[i][j].y[0])||p1.equals(matchedPath[i][j].x[matchedPath[i][j].ver − 1], matchedPath[i][j].y[matchedPath[i][j].ver − 1])) then
                    tmpPList.Add(p2)
                    tmpPList.Add(p1)
                end if
            end if
        end if
    end for
end if
if (bFrequency) then
    DrawMap.drawString(g, AveragePoints[i].Point.x, AveragePoints[i].Point.y, tmpStr2+ tmpStr3 + AveragePoints[i].drawStr, font1, brush2)
else
    DrawMap.drawString(g, AveragePoints[i].Point.x, AveragePoints[i].Point.y, tmpStr3+ AveragePoints[i].drawStr, font1, brush2)
end if
```

1. 各の道路セグメントに対する平均通過速度を取得して、描く座標を生成する（1〜25 行目）
2. 平均通過速度 ($AverageV$) のメッセンジを描く（26〜30 行目）

第 6 章

実験

6.1 実験環境

マップマッチングに使用するデータ地図は「数値地図 25000 さいたま市」を用いた。なお、実データの軌跡の中にこの地図上では存在しない経路を通る場合があったため、実際の経路に沿って部分的にポリラインを追加している。

実験に用いる実データは、GPS ロガーにより自動車の移動を記録した座標データを使用した。設定した時間間隔ごとに、記録した測位点の緯度、経度、記録したときの時間を記録する。

GPS ロガーは GlobalSat 社製の DG-100、BT-335 を使用した。

6.2 GPS ロガーの測位精度

GPS で座標を測定した場合、測距誤差（大気、電離層、受信環境などの要因）により散らばりが生じる。この散らばりの点の平均位置までの距離を二乗平均して平方根をとったものを DRMS といい、平均値の 2 倍、2DRMS が位置精度の誤差の目安とされている。

今回、実験に使用した GPS ロガーの精度はどちらも 10m 2DRMS（$10m$ を半径とする円内に 95% の測位点が入る）である。

6.3 実験結果

実験用パソコン環境

CPU	Intel(R) Core(TM)2 CPU
RAM	4.00GB
OS	Windows Vista

実験用データ

データ地図	数値地図 25000 さいたま市
始点	埼玉正門（502641143, 129318834）
終点	自宅（502639921, 129323194）
GPS 測位サンプル数	1496〜2685
総道路セグメント数	86〜116

実験結果図　図 6.1

実験評価

実験用地図	実験データ	GPS 数量	成功数量	ミス数量	成功率
さいたま市 25,000	09-05-22-21	1131	1131	0	100%

6.4 マッチング精度の比較と評価

マッチング精度の比較　図 6.2, 6.3

実験データ	実行コスト（ミリ秒）	GPS 数量	成功数量	ミス数量	成功率
08-09-06-09	418418.1516	575	510	65	88.7%

実験データ	実行コスト（ミリ秒）	GPS 数量	成功数量	ミス数量	成功率
08-09-06-09	253074.9217	575	575	0	100%

マッチング精度の評価　図 6.4

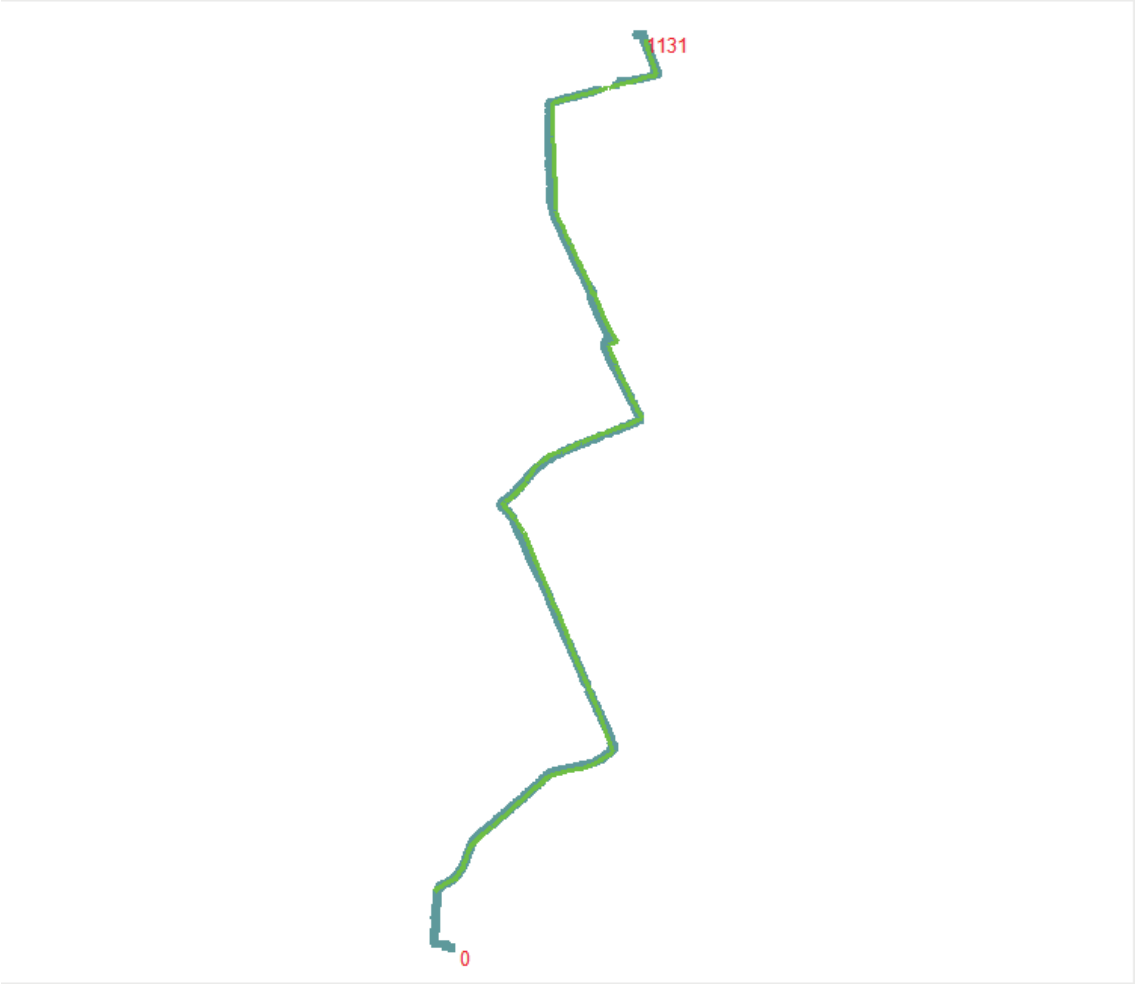


図 6.1: 実験結果

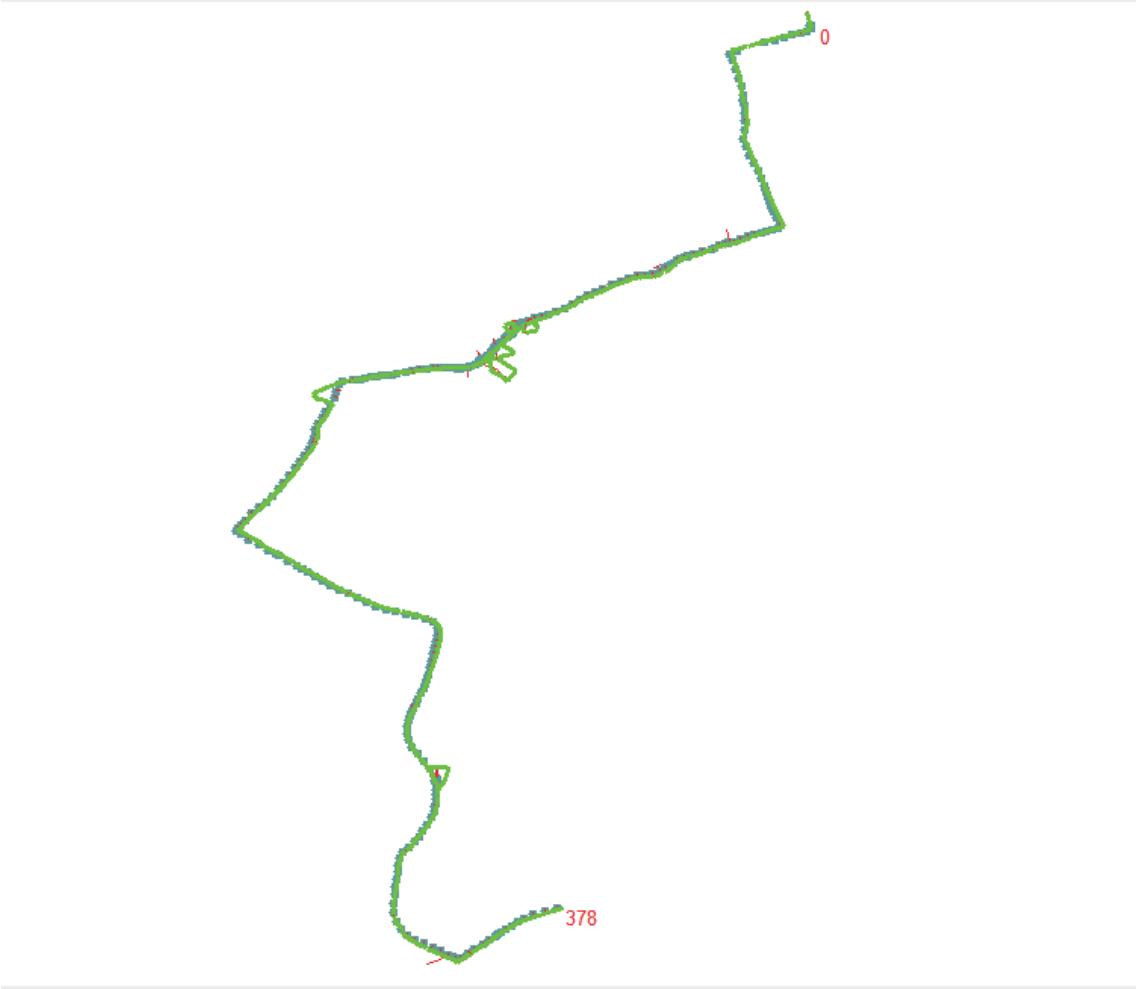


図 6.2: 既存方式の実験結果

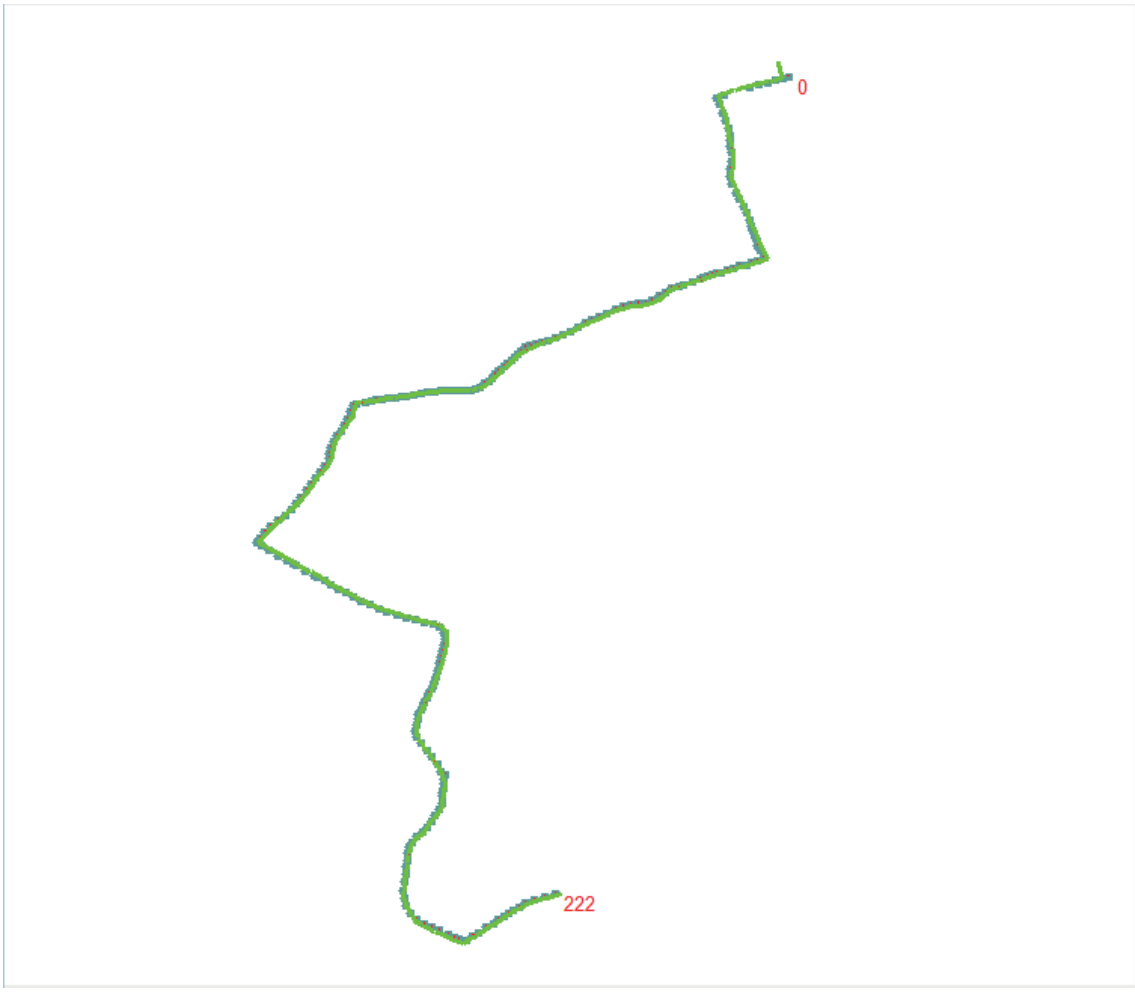


図 6.3: 提案方式の実験結果

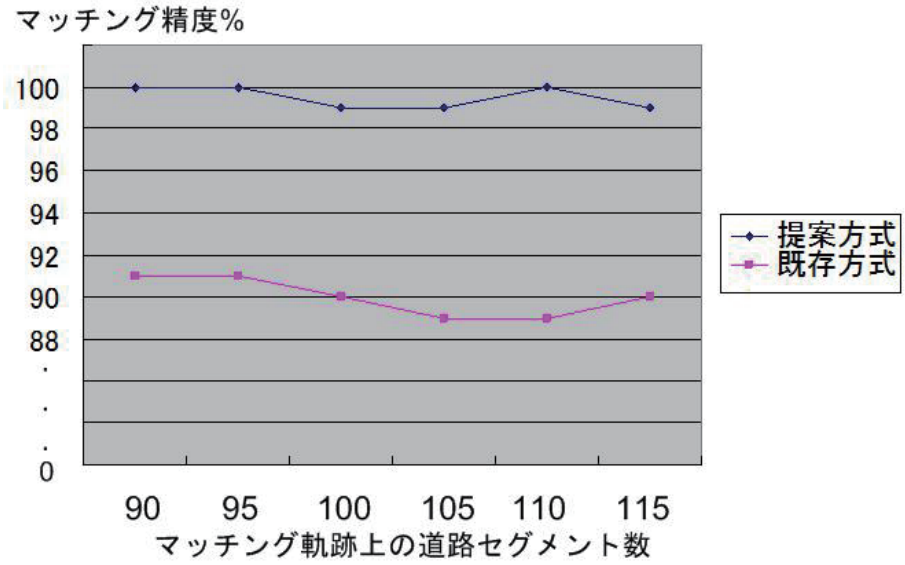


図 6.4: マッチング精度の評価

6.5 マッチング実行時間の比較と評価

マッチング実行時間の比較と評価　図 6.5

6.6 移動体の実時間モニタリングの為のマップマッチングの結果

移動体の実時間モニタリングの為のマップマッチングの結果について、筆者等の一人が自家用車で移動する経路を約 1 年 3ヶ月間にわたって GPS ロガーを用いて計測した。経路の多くは自宅と大学の往復の経路である。

GPS は 1 秒に 1 回の頻度でデータを取得する。このデータを数値地図 25,000 の道路とマップマッチングすることにより、よく通る経路情報を得た。図 6.6 6.7

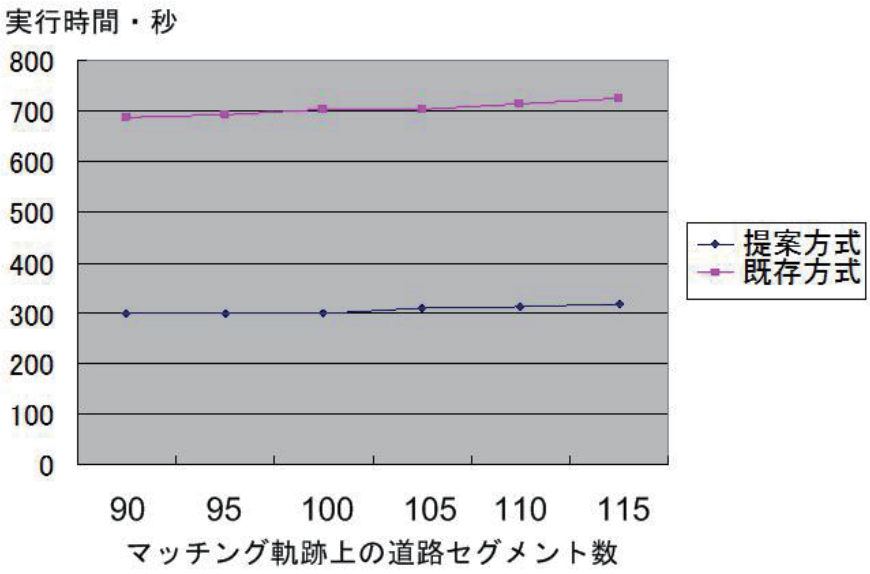
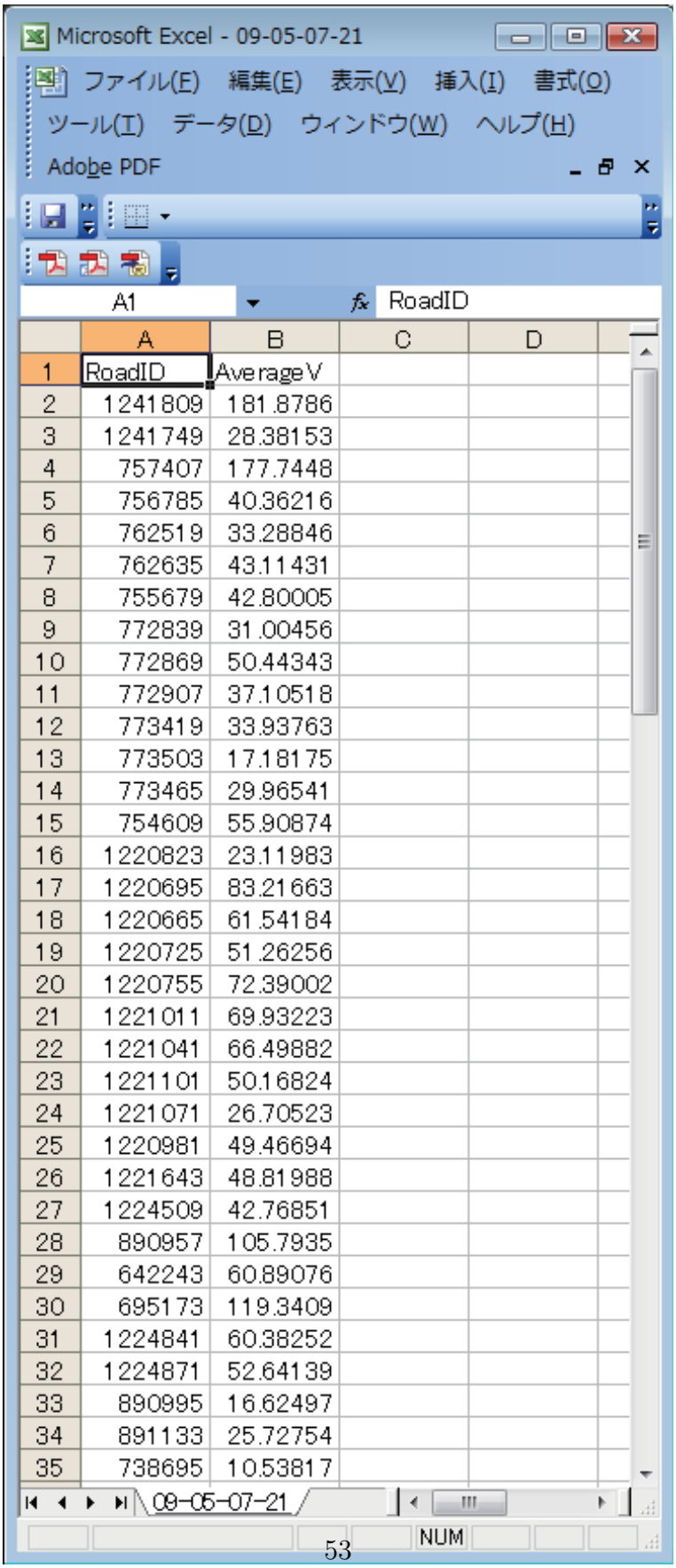


図 6.5: マッチング実行時間の比較と評価

図 6.6: 移動体の実時間モニタリングの為のマップマッチングの結果



	A	B	C	D
1	RoadID	AverageV		
2	1241809	181.8786		
3	1241749	28.38153		
4	757407	177.7448		
5	756785	40.36216		
6	762519	33.28846		
7	762635	43.11431		
8	755679	42.80005		
9	772839	31.00456		
10	772869	50.44343		
11	772907	37.10518		
12	773419	33.93763		
13	773503	17.18175		
14	773465	29.96541		
15	754609	55.90874		
16	1220823	23.11983		
17	1220695	83.21663		
18	1220665	61.54184		
19	1220725	51.26256		
20	1220755	72.39002		
21	1221011	69.93223		
22	1221041	66.49882		
23	1221101	50.16824		
24	1221071	26.70523		
25	1220981	49.46694		
26	1221643	48.81988		
27	1224509	42.76851		
28	890957	105.7935		
29	642243	60.89076		
30	695173	119.3409		
31	1224841	60.38252		
32	1224871	52.64139		
33	890995	16.62497		
34	891133	25.72754		
35	738695	10.53817		

図 6.7: 移動体の実時間モニタリングの為に抽出した「よく通るルート」の情報

第 7 章

まとめと今後の予定

7.1 まとめ

本論文の研究結果は以下の通りになる。

1. 系統的にLBSを研究して、ターミナル相関的な理論と移動ターミナル中のマップマッチング技術に基づいて、マップマッチングに影響する要因を総括した。
2. 既存するマップマッチング方式を研究して、既存方式についてマッチング正確性、マッチング実行コストに関する欠点をまとめた。
3. マッチング正確性を高めるのを目指するため、データ処理について3つの方面の改良を提案した、または、マッチングした初期ポリラインの正確性を保証するため、初期マッチングの改良方式を提案した。
4. 提案したデータ処理アルゴリズムを導入した、マッチング実行コストを高速化できた。
5. 大量のGPSサンプルを用いて提案方式を実験して、実験結果より、提案方式は既存方式よりマッチング正確性の高め及び実行コストの高速化がよく改良になった。
6. 提案方式について、データ処理アルゴリズムでは、移動体時速、サンプル距離間隔及びなす角パラメーターが導入され、異なるGPSサンプルと道路状況も適用できる。
7. 移動体の実時間モニタリングの為のマップマッチング結果を取得した。

本研究では、既存方式のマップマッチングの性能を評価していた、そこでデータ処理及び初期マッチング改良方式に基づく、提案したマッチング性能の評価を行った。既存方式と提案方式の比較実験から、提案方式は高効率であることが示された。

実験結果から、テータ処理及び初期マッチング方式を用いた場合には、マッチング正確性が 6%〜12% 高めるし、実行コストでは 1.65 倍〜2.77 倍の高速化された。

提案方式においてマップマッチングが高効率である理由は、初期マッチング改良では、単純にサンプル始点の方向角度をしたがって初期ポリラインを確定する一面性を免れて、サンプルの初期軌跡の方向角度を参照して、初期マッチングミスを有効的に防止した。

一方、データ処理では、GPS サンプルの時速異常、迂回異常、信号待ち異常、受け取る異常を有効的に取り除いて、マッチング正確性が向上できた。

同時に、データ処理では、元の GPS サンプルデータをこのサンプルの軌跡を保持する前提で、無用のサンプルを削除して、マッチング実行中で、無駄な岐路展開を免れて、マッチングが高速化できた。

7.2 今後の予定

マップマッチングの正確性及び実行コストをいっそう改良する予定である。

マッチング正確性について

1. パラメータの改善
2. データ処理の改善
3. 非常に複雑な道路に対してマッチング方式の検討

マッチング実行コストについて

1. 岐路展開方式の改善
2. 岐路展開数量の改善
3. 異なる道路に対して、岐路展開方式の検討

謝辞

私は大学院に入る前から、電車中に将来の夢としており、日本に結婚して就職したいと思っていました。そのため社会で働きたいという思いも早くから抱いております。

私は 3 歳からピアノの練習が始まり、そのとき一番嫌いことを思っていました。いつも考えていました、なぜほかの子供が自由に遊べ、自分がピアノに向き合わないといけなかったの。23 後の現在、自分が皆さんの前に楽曲を自由に弾くときに、答えが出てきました。23 年間の練習は私に音楽の芸術修養を上げ、また成功がほしいならば、努力は必要として道理がわからせました。

私はすべての中国の子供さんと同様にそのまま成長していました。いつも家族からお世話をもらっていますし、すべてを当たり前と考えていました。自分がいつも生活費、交通料、学費について心配してなかったです。この状態が続いて、私は英語の専門学校にはいて、大学に入学して、いつも努力に勉強して英語と機械構造専門二つの卒業証書をもらいました。自分は英語の自信がありますし、英語で交流とか書くとか問題がないと思います。でも当時にぜんぜん気がついてなかったですが、自分の社会人になる準備にならなかったです。

来日した後に、ぜんぜんわからない世界に対して、どうしたらよいかわからなかったです。そうどころか、完全的な開始が始まりました。

どんどん日本が少し了解になっていました。自分の生活責任も負担していました。研究室の仕事と研究をしたら、就職活動をすると、社会人になる磨きです。毎回自分の労働によって生活していると思って、社会人になる目標に接近しています。毎回研究室のミーティング及び反省会を終わったら、社会人引き受ける責任感をわかります。

日本に住んでいる 3 年間、いろいろなことを理解しました。今の私は、努力、向上、確固たる信念を持って、責任感を持って、芸術修養も持っている留学生です。日本と上海にいる家族、先生と友達、自分の努力、特に日本社会自体におかげです

べての変化は可能です。

研究ならびに生活面において、多大なる助言やご指導を賜りました大沢　裕教授には心から感謝いたします。大沢先生からの指導におかげさまで、自分の短所及び能力をもう一回認識しました。大沢先生からの指導におかげさまで、今回卒業が可能になりました。また、川崎　洋准教授にも多くの手助けをしていただきましたこと深く感謝いたします。

また，先輩としていつも良きアドバイスをいただきました、高沢　聡氏、大木　彩加氏、栗原　孝暢氏、そして、大沢研の皆様、伊藤　雅亮氏、油井　真斗氏、藤井　和史氏。大変お世話になった皆様、福沢　由之氏、路　毅氏、耿　輝氏。並びに私を暖かく見守って頂いた両親、祖父母をはじめとする周囲のすべての皆様に深く感謝いたします。

参考文献

[1] French R L. Map matching origins approaches and applications. In *Proc.of Second International Symposium on Land Vehicle Navigation*, pp. 99–116, 1989.

[2] Zhao Yilin. Vehicle location and navigation system. In *Artech House*, pp. 83–103, 1997.

[3] MEEL Major and Ph Bonnifait. A roadmap matching method for precise vehicle localization using belief theory and kalman filtering. In *Proe.of International Conference on Advanced Robotics*, pp. 1677–1682, 2003.

[4] Dakai Yang, Baigen Cai, and Yifang Yuan. An improved map-matching algorithm used in vehicle navigation system. In *Proc.of IEEE Intelligent Transportation Systems*, pp. 1246–1250, 2003.

[5] Abbott E and Powell D. Land vehicle navigation using gps. In *Proccedings of THE IEEE*, pp. 145–162, 1999.

[6] R Kuehne, R-P Schaefer, J Mikat, K-U Thiessenhusen, U Boettger, and S Lorkowski. New approaches for traffic management in metropolitan areas. In *Proc. IFAC CTS Symposium*, 2003.

[7] S Brakatsoulas, D Pfoser, and N Tryfona. Practical data management techniques for vehicle tracking data. In *Proc. 21st ICDE conf*, pp. 324–325, 2005.

[8] D Pfoser and C S Jensen. Capturing the uncertainty of moving-object representations. In *In Proc. 6th SSD conf*, pp. 111–132, 1999.

[9] Ouri Wolfson, Sam Chamberlain, Son Dao, Liqin Jiang, and Gisela Mendez. Cost and imprecision in modeling the position of moving objects. In *Proc. ICDE*, pp. 588–596, 1998.

[10] Ouri Wolfson, A PRasad Sistla, Sam Chamberlain, and Yelena Yesha. Updating and querying databases that track mobile units. In *Distributed and Parallel Databases*, pp. 257–287, 1999.

[11] Ouri Wolfson, Bo Xu, Sam Chamberlain, and Liqin Jiang. Moving objects databases: Issues and solutions. In *In Proc SSDBM*, pp. 111–122, 1998.

[12] David Bernstein and Alain Kornhauser. An introduction to map matching for personal navigation assistants. In *Technical report, New Jersey TIDE Center*, 1996.

[13] Sotris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *In Proc. 31st VLDB*, pp. 853–864, 2005.

[14] Joshua S and Greenfeld. Matching gps observations to loations on digital map. In *In In Proc. 81th Annual Meeting of the Transportation Reseach Board*, 2002.

[15] Mohammed A. Quddus, Washington Yotto Ochieng, Lin Zhao, and Robert B. Noland. A general map matching algorithm for transport telematics applications. In *GPS Solutions*, pp. 157–167, 2003.

[16] Huabei Yin and Ouri Wolfson. A weightbased map matching method in moving objects databases. In *Proc. 16th SSDBM*, pp. 437–438, 2004.

[17] Dimitris Papadias, Jun Zhang, Nikos Mamoulis, and Yufei Tao. Query processing in spatial network databases. In *Proceedings of the 29th VLDB Conference*, pp. 790–801, 2003.

[18] Elias Frentzos. Indexing objects moving on fixed networks. In *Proc. SSTD*, pp. 289–305, 2003.

[19] Zhiming Ding and Ralf Hartmut G uting. Managing moving objects on dynamic transportation network. In *Proc. 16th SSDBM*, pp. 287–296, 2004.

[20] Alminas Civilis, Christian S, Jovita Nenortaite Jensen, and Stardas Pakalnis. Effcient tracking of moving objects with precision guarantees. In *Technical Report TR-5, Department of Computer Science, Aalborg University*, 2004.

[21] Alminas Civilis Christian S. Jensen and Stardas Pakalnis. Techniques for e cient road-networkbased tracking of moving objects. In *IEEE Trans. on Knowledge and Data Engineering*, pp. 698–712, 2005.

[22] Wei-Shinn Ku, Roger Zimmermann, Haojun Wang, and Chi-Ngai Wan. Adaptive nearest neighbor queries in travel time network. In *ACM GIS'05*, 2005.

[23] Kyriakos Mouratidis, Man Lung Yiu, Dimitris Papadias, and Nikos Mamoulis. Continuous nearest neighbor monitoring in road networks. In *Proc. 32th VLDB*, pp. 43–54, 2006.

[24] Yu-Ling Hsueh, Roger Zimmermann, Haojun Wang, and Wei-Shinn Ku. Partition-based lazy updates for continuous queries over moving objects. In *ACM GIS'07*, 2007.

[25] YDalia Tiesyte and Christian S. Jensen. Effcient cost-based tracking of scheduled viecle journeys. In *The Ninth International Conference on Mobile Data Management*, pp. 9–16, 2008.

[26] YDalia Tiesyte and Christian S. Jensen. Similarity-based prediction of travel times for vehicles traveling on known routes. In *ACM GIS'08*, 2008.

[27] J. Greenfeld. Matching gps observations to locations on a digital map. In *Proc. 81th Annual Meeting of the Transportation Research Board, Washington, DC*, 2002.

[28] F. van Diggelen. Gps accuracy: Lies, damn lies, and statistics. In *GPS World*, pp. 41–45, 1998.