A Map-matching Algorithm to Improve Vehicle Tracking Systems Accuracy

Agung Dewandaru
Universiti Teknologi Petronas

June 25, 2008

UNIVERSITI TEKNOLOGI PETRONAS

Approval by Supervisor(s)

The undersigned certify that they have read, and recommended to the Postgraduate Studies Programme for acceptance, a thesis entitled "A Map-matching Algorithm to Improve Vehicle Tracking Systems Accuracy" submitted by (Agung Dewandaru) for the fulfillment of the requirements for the degree of (Master of Science in Information Technology)

Date

Signature : _____

Main supervisor : ABAS MD SAID

Date : 25/6/08

co-Supervisor 1 : _____

TITLE PAGE

UNIVERSITI TEKNOLOGI PETRONAS

A Map-matching Algorithm to Improve Vehicle Tracking Systems Accuracy

By

Agung Dewandaru

A THESIS

SUBMITTED TO THE POSTGRADUATE STUDIES

PROGRAMME

AS A REQUIREMENT FOR THE

DEGREE OF MASTER OF SCIENCE

IN INFORMATION TECHNOLOGY

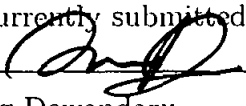
BANDAR SERI ISKANDAR

PERAK

JUNE, 2008

Declaration

I hereby declare that the thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTP or other institutions.

Signature : 

Name : Agung Dewandaru

Date :

Acknowledgement

First of all, the praise should be for Allah for all His grace and bounty. For one of His most important gift to mankind is their curiosity through things, effectively flowering the science and advance the humanity.

Many thanks should be addressed to Universiti Teknologi Petronas for giving me the opportunity to present a very fruitful research work. I would especially give my appreciation to my thesis advisors, Dr Abas and Dr Nasir for patiently guide me for this lengthy thesis work and for their challenging, clear-cut research objectives. Last, thanks to all my family, especially Nina and Ibrahim for all their supports which are very vital to the completion of this work.

Abstract

The satellite-based vehicle tracking systems accuracy can be improved by augmenting the positional information using road network data, in a process known as map-matching. Map-matching algorithms attempt to estimate vehicle route and location in a particular road map (or any restricting track such as rails, etc), in spite of the digital map errors and GPS inaccuracies. Point-to-curve map-matching is not fully suitable to the problem since it ignores any historical data and often gives inaccurate, unstable, jumping results. The better curve-to-curve matching approach consider the road connectivity and measure the curve similarity between the track and the possible road path (hypotheses), but mostly does not have any way to manage multiple route hypotheses which have varying degree of similarity over time. The thesis presents a new distance metric for curve-to-curve map-matching technique, integrated with a framework algorithm which is able to maintain many possible route hypotheses and pick the most likely hypothesis at a time, enabling future corrections if necessary, therefore providing intelligent guesses with considerable accuracy. A simulator is developed as a test bed for the proposed algorithm for various scenarios, including the field experiment using Garmin e-Trex GPS Receiver. The results showed that the proposed algorithm is able to improve the map-matching accuracy as compared to the point-to-curve algorithm.

Keywords: map-matching, vehicle tracking systems, Multiple Hypotheses Technique, Global Positioning System.

Abstrak

Kejituan Sistem Pengesan Kenderaan berdasarkan satelit boleh dipertingkatkan dengan memperlengkapkan informasi posisi memakai data jejaring jalan, dalam sebuah proses yang dikenali sebagai map-matching. Algoritma map-matching mencuba memperkirakan jalan dan lokasi pada peta jalan, walaupun ketidakjituan pada peta dijital dan GPS. Map-matching jenis titik-ke-kurva tak sepenuhnya tepat diterapkan karena mengabaikan data sejarah dan seringkali memberikan hasil yang tidak jitu, tidak stabil, dan melompat. Pendekatan kurva-ke-kurva lebih baik karena mempertimbangkan konektifitas dan mengukur kemiripan kurva antara track kenderaan dan rute jalan yang mungkin (hipotesis), namun tidak memenej beberapa hipotesis yang memiliki darjah kemiripan kurva yang berubah-ubah. Thesis ini mengandungi sebuah metrik jarak untuk teknik map-matching kurva-ke-kurva, digabungkan dengan sebuah framework algoritma yang boleh mempergunakan banyak hipotesis rute dan memilih hipotesis paling tepat pada satu saat, dan membolehkan pembetulan pada masa hadapan bila diperlukan, sehingga menghasilkan tebakan cerdas dengan kejituan yang baik. Sebuah simulator telah dibangun sebagai test bed untuk algoritma ini dengan bermacam senario, termasuk ujikaji di lapangan memakai receiver GPS Garmin e-Trex. Hasilnya menunjukkan bahwa algoritma yang dicadangkan boleh mempertingkatkan kejituan map-matching apabila dibandingkan dengan algoritma titik-ke-kurva.

Kata kunci: map-matching, Sistem Pengesan Kendaraan, Multiple Hypotheses Technique, Sistem Kedudukan Sejagat (GPS).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The demand and application of vehicle tracking system is rapidly increasing, due to its usefulness in terms of commercial or personal aspects, and also due to the availability of Global Navigation Satellite System (GNSS) such as GPS which had started in 1995. There is a broad range of applications of the vehicle tracking system, such as in-car navigation systems, advanced traveler information, dynamic route guidance systems, fleet management, collision avoidance systems, or even emergency rescue technology assisted by such system. The technologies involved are also varying from the use of satellite, radio communication link, RFID tag/sensors, gyrometer, odometer, and also the use of camera to provide data for scene analysis.

These Vehicle Tracking Systems applications typically require continuous and accurate positioning information on the vehicle traveling on the road network. Many of them also need real-time display of the vehicle location on road in a map with a great accuracy [1]. In order to achieve that, the system needs to use the available digital road map data and to integrate that with the estimate locational data of the vehicle provided by the GNSS. This process is called map-matching which is based on a particular algorithm, and typically has assumption that the vehicle always takes place on one link of the road network.

Map-matching (and its visual user interface) has also become one goal to be achieved in many navigation systems, for its user-friendliness as compared to bare display of coordinates. It is important not only for navigation and map display, but also for advanced driving assistance system (ADAS) applications such as adaptive cruise control, adaptive lighting control, lane departure warning, and transmission shift control. Map-matching is also required for the ultimate goal of intelligent vehicle research: autonomous driving [2].

Unfortunately, map-matching inaccuracy is still an obstacle for these applications. The challenge of map-matching algorithms lies in that it has to reconcile inaccurate locational data with inaccurate digital road network (map) data into an accurate estimation

of the road segments traveled by the vehicle. If both digital maps and vehicle location estimate are perfectly accurate, the algorithm would be a straightforward projection of a point in the map road system [3]. Unfortunately, this is almost always not the case. GPS readings vary over time, and it degrades when the environment is rather inconducive, such as in urban canyons, roads with dense tree cover, or a tunnel [1]. Moreover, the map is also subject to many error factors, complicated with the fact that the true location of the vehicle is not always lie in the one dimensional road center-lines, but can be anywhere "inside" the two dimensional road surface.

The map-matching inaccuracies will be seen as an off-road, or erroneous projection of the vehicle on the road map, i.e. the vehicle is reported on the wrong road segment. This *spatial mismatch* phenomenon happens more often at junctions, roundabouts, complicated flyovers [1] where there are many plausible road segment alternatives, or in such areas where the GPS degrades as mentioned before. Of course such inaccuracies will decrease the usefulness of any implementation of a vehicle tracking systems.

Therefore, our objective is to develop a map-matching algorithm which is able to give improvement on the accuracy in the field of vehicle tracking systems. The accuracy will be measured by a proportion of correct road matchings and the number of matchings made. A matching is correct if the vehicle is indeed located on the matched road segment. In other words, the algorithm must be able to present estimate location which coincides the road system (or subway system etc) [3]. The algorithm must also able to present unambiguous, meaningful travel route [4]. The algorithm shall aim at real-time usage, effectively enabling application of in-vehicle navigation and many services which requires it. This thesis will assume the use of GPS as the primary data source, while still acknowledging the potential use of other GNSS (currently GLONASS or Galileo). Hence, it is assumed that GPS characteristics could represent GNSS in the scope of map-matching.

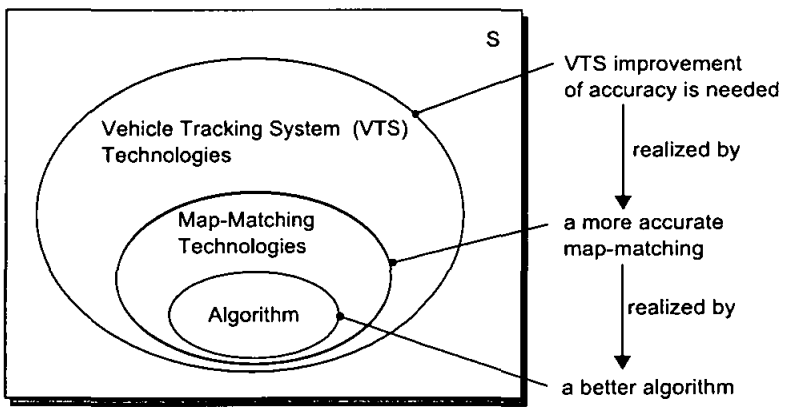


Figure 1.1: Research Area

The thesis provides sufficient background in the Chapter 2, then proceeds to discuss the cutting edge algorithms to accomplish map-matching in Chapter 3. A new distance metric is considered necessary and the development of it is presented in Chapter 4. Chapter 5 discuss about the development a framework algorithm for the distance metric mentioned. Both algorithm will be implemented and tested by a map-matching simulator, presented in Chapter 6. The proof of concept of the theoretical algorithm was given by the experiment, which is explained in Chapter 7. The discussion and conclusion will compose the rest of the chapters (Chapter 8 and Chapter 9).

Chapter 2

Background

2.1 Measuring Accuracy

This thesis carries the objective to improve the accuracy of a Vehicle Tracking Systems, which justifies for a discussion of the specific meaning of "accuracy". Engineers often define accuracy as the degree of perfection in measurement which denotes how close a given measurement is to the true value of the quantity [5]. This definition assumes that the actual value of a scalar variable will never be known exactly, and it is approached through the measuring process. Therefore, according to this definition, there is no such perfect measurement, and accuracy is often stated using some certainty and error factor. The example of such statement is like, "The value is $50.1m$ with a 95% chance that this measurement has $\pm 0.2m$ error".

There is another meaning of accuracy [6, 7] which is widely used in the pattern recognition field which will be better representing exact result experiment. In this meaning, the accuracy is used to characterize a particular classifier in its ability to correctly classify an object. Accuracy is represented as the proportion of correct classification to total number of classification. Using this measure, a perfect (100%) accuracy in an experiment is achievable, as long as it satisfies the definition of correct (and incorrect) result. Because the nature of map-matching is to infer the estimate road segment, then its outcome could be classified into a correct and incorrect result. Such similar classification is also used in [8]. This latter definition suits the problem of map-matching better, and will be used throughout this work.

2.2 Global Positioning System

Global Positioning System (GPS) is a satellite-based radio navigation system owned by US Government. It is able to give positional, speed and heading information to all its users concurrently. It started its full operational capability in 1995 with 24 satellites, providing public use with horizontal positional accuracy to up to 100 m. With the Selective Availability (some kind of degrading mechanism, so that GPS gives lower accuracy for public, non-military use) turned off by the US Government recently, a significant improvement of accuracy was gained for most users, achieving accuracy within 15 m for 95% of time.

GPS provides continuous positioning and timing information anywhere in the world under any weather conditions. Also, it provides the heading (direction) and speed information of the receiver. All the communication is done one way from the satellite to the receiver, so the calculation of multiple GPS signal in the receiver's side is needed to obtain those information.

2.2.1 GPS error factors

The GPS itself has a list of error sources, which could be divided into three large segments: satellite-related, receiver-related, and atmospheric errors and biases. The error estimation for each item is elaborated in Table 2.1 adapted from [9]. In the table, two groups of columns are listed: The C/A code and P(Y) code. These two are the codes transmitted by GPS satellites signal, referring to two level of service, SPS (Standard Positioning Service, for public use) and PPS (Precise Positioning Service, for military use), respectively.

Error Source	C/A	P(Y)
Satellite Clock and Ephemeris Error	3.9	3.9
Ionospheric Delay	9.9	3.1
Tropospheric Delay	2.9	2.0
Receiver Noise and Resolution	11.1	1.1
Multipath	12.6	1.2
Selective Availability (if present)	30.0	N/A

Table 2.1: GPS Error Sources and Magnitude (in meter)

Urban Canyon GPS problems

In the situation where there are many tall buildings or obstacles to block the satellite signal (blocked line of sight as shown in the left part of Figure 2.2.1), the GPS accuracy

may degrade significantly. This is partly because the GPS needs as many satellite signal as possible to produce more accurate result. Another problem is because that of multipath error (the right part of Figure 2.2.1), whereby a reflected satellite signal (e.g.: reflected by building) has lower quality and interfere the original signal, thus degrading the overall accuracy. This problem is commonly found in the urban area setting (thus called urban "canyon" problem).

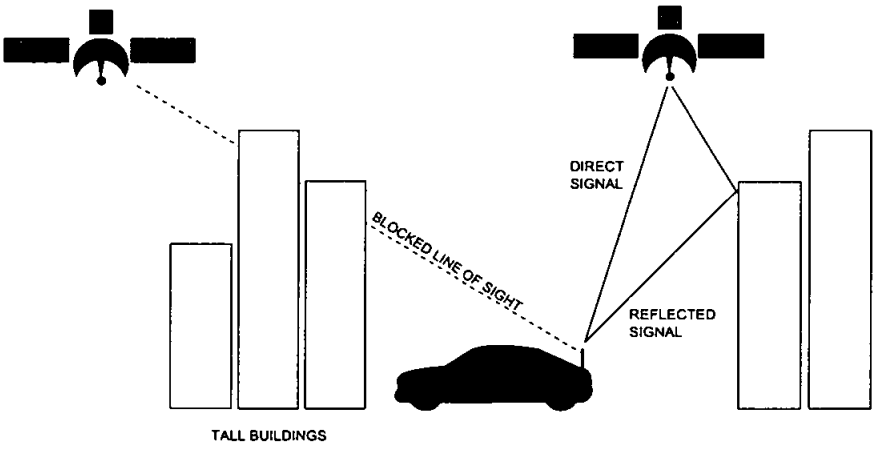


Figure 2.1: Blocked Satellite LOS and Multipath Problem

Lower Heading Accuracy on Lower Speed

One of the observed characteristic of GPS is the decline in heading accuracy when the receiver is on a low speed, also noted by [3, 1]. When the vehicle stops, the heading information accuracy gets even worse. While this could be improved by using more than one antenna and a special search algorithm such the one presented in [10, 11], the usage of GPS heading especially in the junction might not be reliable.

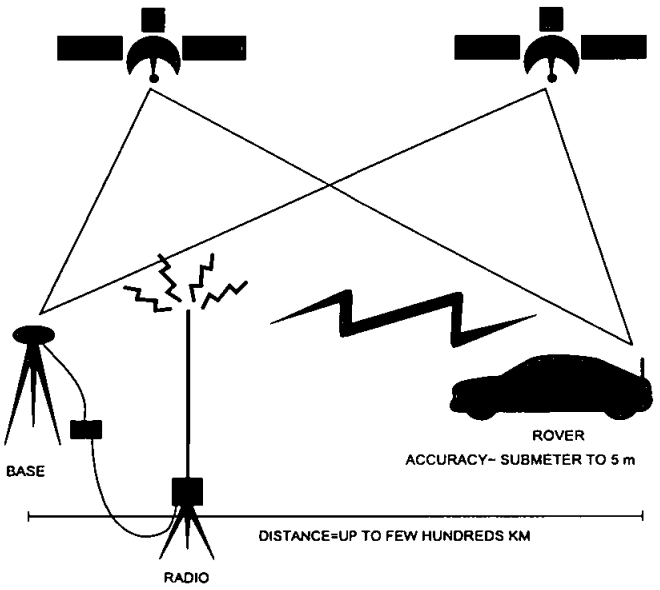
2.2.2 Minimizing GPS error

Differencing techniques such as Differential GPS (DGPS) and Real Time Kinematic (RTK) are available to improve the accuracy, typically by monitoring the data from two GPS receiver in some distance. The two sources of the data collected will roughly share the same satellite and atmospheric errors, which could be used to reduce the errors by differencing and performing correction using the data [12]. Figure 2.2a and Figure 2.2b shows the architecture for a double differencing technique for DGPS and RTK.

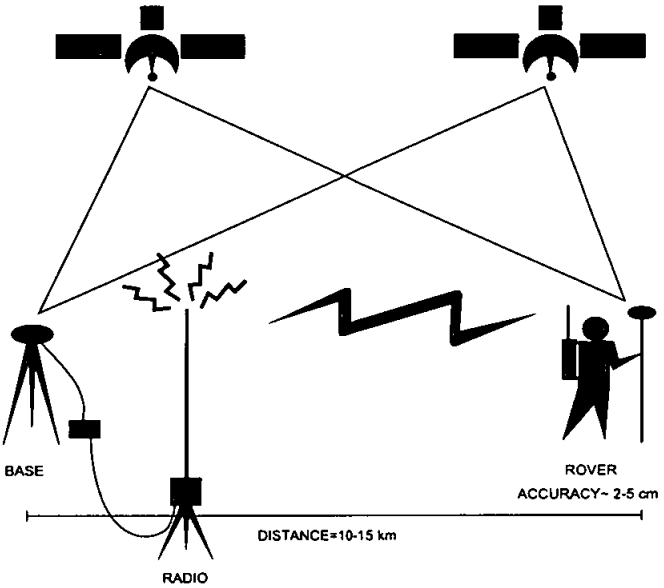
Some works on map-matching (as an example the work of [13]) are using differencing technique to approximate the "true" value, in order to measure accuracy of their map-

matching techniques, thereby providing the standard deviation and the error mean of the reported location. While this may potentially be useful to see the general trend, this method ignores the basic question in map-matching, i.e. whether the matching correctly finds the road the vehicle is traveling. For example, assume that a road map has an error for 10-20 meters. A good map-matching algorithm with a very high accuracy may be reported as having a large error mean value (due to the map error), even though it actually does a good performance inference by correctly outputting the entire road traveled by the vehicle.

Although DGPS and RTK could provide much improvement on accuracy to up to (1-5 m for DGPS, centimeter accuracy for RTK), the application of this technique requires additional base receiver which will calculate the correction. Moreover, some kind of communication network infrastructure (typically terrestrial radio) is needed to integrate this correction [12]. These provisions might still be considered costly for most situations worldwide, therefore basic map-matching without any of these support is still necessary.



(a) Differential GPS



(b) Real Time Kinematic

Figure 2.2: Example of Differencing Techniques

2.2.3 Augmenting GPS: Dead reckoning and Inertial Navigation System

Dead reckoning (DR) technique is a position estimation method based on the idea that the current position can be derived from the earlier position provided that the heading and distance traveled is known.

The DR system is typically comprised of odometer sensor and a vibration gyroscope. The travel distance is obtained from odometer, while the gyroscope measures the vehicle heading. The odometer works by counting the number of revolution of vehicle wheel multiplied by a calibrated scale error factor. This way the distance that the vehicle has traveled could be obtained. The odometer's scale error factor will accumulate rapidly, causing significant positional error if left uncompensated [12].

Vibration gyroscope works by measuring voltage change in the vibration gyro which is proportional to the angular velocity of the vehicle. It then multiplied by a scale-factor to obtain the heading rate. The gyroscope is sensitive to the temperature (gyro bias) and to the gyro scale-factor error when taking a turn. This make the gyroscope is also subject to accumulative error just like odometer [12].

An Inertial Navigation System (INS) is a navigation system which relies on the initial position, velocity, and attitude (orientation), and thereafter measures the attitude rates and accelerations. It is the only form of navigation that does not rely on external references [9].

The characters of those two systems (DR and INS) are complementary to that of GPS: the accuracy get worse over time, but it give a good short term accuracy. The typical GPS setting, on the other side, may not give such good short-term accuracy, but the accuracy does not degrade over time. This traits make a good combination of GPS/INS or GPS/DR, usually integrated together by employing Kalman filter as the integrator.

2.2.4 Application of GPS in vehicle tracking systems

In many of the available vehicle tracking systems, GPS is generally used for vehicle navigation as the primary and initial data source. To improve its accuracy, it may be combined with DR technique as explained above. Besides DR technique, people have been using signpost technology and terrestrial radio navigation system. The use of these multiple data sources might help to correct the error on the GPS (or other satellite navigation system) position output [14].

2.3 On Mapping

A map is a model of geographical features, typically carrying relevant information to serve some explanatory purpose which could not be easily understandable otherwise. Map of continents, as an example, is a reduction and projection of a very large objects (the continent itself) so that man could easily grasp the real object without having to fly up into the upper atmosphere to see it [15].

2.3.1 Map Projection

A map is usually displayed in a two dimensional medium, whereby a kind of map projection must be used. Map projection is a transformation that distort a three dimensional object into two dimensional image. Thus, there always exist inherent inaccuracies on every distortion. Further, there is always something sacrificed in terms of preserving other goal. For example, the conformal projection sacrifices the area information of a land parcel for the sake of keeping direction intact. In contrast, the equal-area projection works by maintaining the area while sacrifices the direction [16].

Maps creation is greatly helped by the remote sensing techniques such as aerial photographs, radar or satellite images. Google Earth is a service which is based on satellite imagery to generate the aerial image data for the experiment of this research work. According to [17], Google earth uses Simple Cylindrical Projection for its imagery base. The cylindrical projection is considered conformal and it is easy to convert into a two dimensional plane, cartesian coordinate.

2.3.2 Datum

Datum is a reference network consisted of a reference point and a spheroid to model the real earth (i.e., the geoid). It has information on the estimate land height of any requested earth coordinate (latitude and longitude). GPS uses the WGS-84 datum, and the standard GPS receiver units are defaulting to this datum. This means that all reported coordinates are relative to that particular datum. Two map information that uses different datum cannot be used together unless some conversion is applied. Google Earth, the map base that we use, is also servicing in respect to WGS-84, and thus compatible with the GPS reported locations [17].

2.3.3 Mapping error

There are some errors typically involved in (digital) map creation:

1. Digitizing error

Digitizing is the process of entering nodes and vertices that represents the map features(could be road, area etc). It could be done using puck or using software (on screen digitizing), where the human factors involvement is prevalent. The error could also come from the simplification of features. For example. a curving road that is not straight may be modeled using sequence of nodes only.

2. Georeferencing and rectification error

GIS data usually must have a real world coordinate system (such as latitude-longitude). Georeferencing is a process of registering, or fixing, data to a standard coordinate system, thereby linking the map to the earth. The best method of establishing a proper georeference is to define at least four reference points (sometimes called tic points) around the area being digitized (close to the corners if possible), each with a precisely known real world coordinate position that is typed into the program. Only with some known reference points, digitized features can be properly located on earth. After georeferencing, the map rectification could be made, by adjusting the image using affine transformation that stretches and deforms it according to the reference points [18]. After the rectification, normally each reference points normally have zero error, except if there are more than three reference points. For a three reference points an exact mathematical transformation could be calculated so that all points has the same coordinate with the already specified coordinates and thus have zero error.

2.4 Map-matching

As already stated in the Introduction chapter, map-matching is a process to integrate locational data of the vehicle with the digital map road data, to improve the accuracy and also to give meaningful information to the vehicle tracking systems user.

Map-matching procedures have many approaches, varying from simple point-to-point search to the use of more complex statistical techniques such as integration using Kalman Filters as in [19]. It is noted that the map-matching problem is complex and fairly difficult task [20] [3] since there are many error factors involved. Hence, the simple and naive point-to-point and point-to-curve is unlikely to work very well, so more sophisticated algorithm must be used.

Before further elaboration on the various algorithms in Chapter 3, the map-matching formal problem statement adapted from [3] is included: The concern is about a vehicle (might be abstracted to any agent) moving along a finite system (or set) of roads (or,

a more general concept, tracks, such as railways), $\overline{\mathcal{N}}$. At each time period $T \in \mathbb{N}_0$, the system provides the estimate of the vehicle's location (which is usually obtained from satellite navigation system). The actual location is denoted by $\overline{P}$ and the estimate is denoted by P^t. The goal of map matching is to determine the road in $\overline{\mathcal{N}}$ which contains $\overline{P}^t$. The "true" road system, $\overline{\mathcal{N}}$ is rather unknown exactly, instead, there is *network representation*, $\mathcal{N}$, consisting a set of curves in $\mathbb{R}^2$, each of which is called an *arc*. Each arc is assumed to be piece-wise linear, and arc $A \in \mathcal{N}$ can be characterized as a finite sequence of points $(A^0, A^1, ..., A^{nA})$, each point $A^n \in \mathbb{R}^2$. The A^0 and A^{nA} are called the nodes, where it may represents a connection to other arc. The rest of the points A^n in the set are referred as *shape points*, and between any point within any arc A^k and $A^{k+1}, 0 \leq k < nA$ there is an edge called *arc segment* connecting the two points. The problem is called map-matching problem because the first goal is to match the estimated location P^t, with an arc $A \in \mathcal{N}$, and then determine the road, $\overline{A} \in \overline{\mathcal{N}}$, which corresponds to the person's actual location $\overline{P}^t$. The secondary goal is to infer on which spot of the road the vehicle is in. That is, a map-matched (or snapped) point $\overrightarrow{P}^t$ on A that best corresponds to $\overline{P}^t$. The third goal of map-matching is delivering a set of road arcs that form an *unambiguous, meaningful travel route* of the vehicle. That is, providing a connected sequence of $(\overline{A}_0, \overline{A}_1, \overline{A}_{...}, \overline{A}_k)$ which resulted from set of A resulting from the first goal.

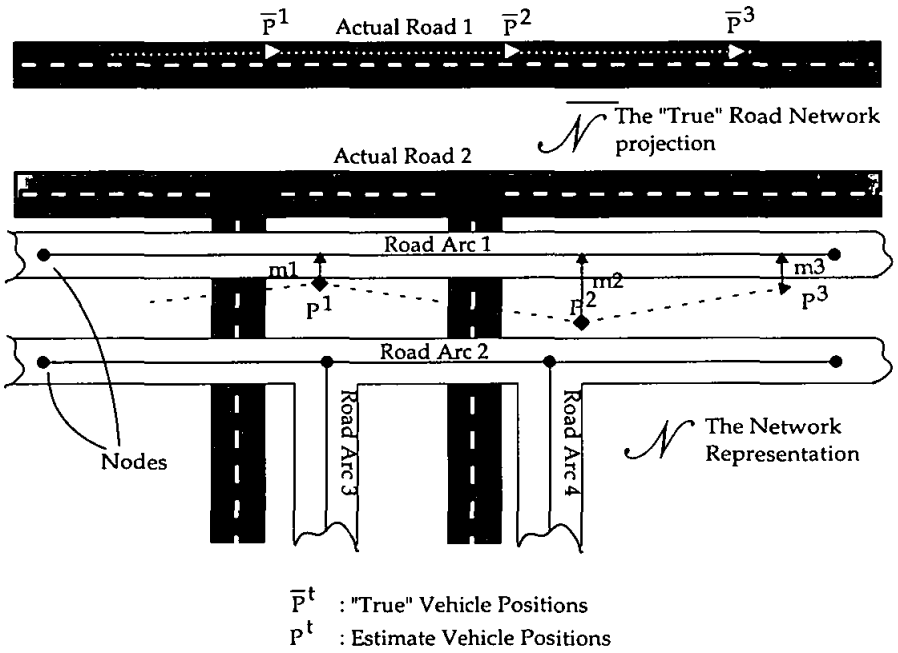


Figure 2.3: The Problem of Map-matching

The illustration on the problem can be seen in Figure 2.3. Assume we have the true, exact road network information and projected on the figure as a aerial figure of road lane. The network representation however, is not perfectly accurate. Hence, it differs from the exact road. It is modeled using a set of arcs, $\mathcal{A} = \{A_1, A_2, A_3, A_4, A_5, A_6\}$ (see Table 2.2). Each arc is composed of two nodes and a number of shape points. In this example each arc is comprised from only two nodes and there are no intermediate shape points. Every points on this map have specific coordinate according to the coordinate system used. If Node 1 is expressed in (0.0,0.0) then Node 2 might be expressed in (550.0,80.0) and Node 3 is (0.0,80.0).

Arc	Nodes
A_1	(1,2)
A_2	(3,4)
A_3	(4,5)
A_4	(5,6)
A_5	(4,7)
A_6	(5,8)

Table 2.2: A network representation example

The term *arc* and *curve* used on the definition actually have the same semantic. Both are used to refer to a polyline or piecewise linear curve, i.e. curve composed from a sequence of points. Throughout this thesis the term *arc* which carries the (road) topology notion will be used in referring the network representation, while the term *curve* will be used more often in the discussion of distance metric to measure its similarity.

Note that extra information such as heading, speed, and road width is not used on this definition of map-matching. Such information might be valuable to further improve the accuracy, but is not mandatory, and could be incorporated later after the first prototype had been built. The approach used here is to start from the simple but solid model definition, as already described in this section.

2.5 Parametrizing Curves

It is often helpful to model a curve A as a parametrized function $a(t)$, which is a vector-valued function of a real variable [21]. It means that the function returns a vector that represent a target that moves along the curve. In two dimensional xy-plane, $a(t)$ returns a pair of x,y values (coordinate) indicating a point within the curve at parameter t. This t has the range $[0, 1]$, so $a(0)$ would mean the initial point of curve A and similarly $a(1)$ mean the end-point of curve A. Since we use a piecewise-linear curve to approximate the real curve, the coordinate obtained from the function would always coincide one of the line constructing the curve.

2.6 Algorithm and Data Structure

This section will provide a basic of the algorithms and data structures which will be used extensively throughout this thesis.

2.6.1 List and Linked List

A list is an ordered set of item, and might be implemented as a linked list, especially when the dynamic collection of objects is needed. Every element in a (single) linked list has the data item to be stored and a pointer (memory address) of the next node. Thus a linked list does not need to be stored contiguously in memory [22]. Among the operation that is commonly defined is adding the last element and deletion (clearing) of all element. Note that if we attach object reference (or pointer) as the data item within the node, then the list clearing will not deallocate the object from memory.

A list, however, might be implemented as an array, in which there is substantial work of shifting the nodes in case of node deletion in the middle of the list. The developed simulator program(see Chapter 6) is using this version of list. However, this difference is not significant for further discussion, and both implementation is fine as long as there are some basic primitives for adding element, deleting, and clearing all element from a list.

2.6.2 Stack

Stack is a data type that has LIFO (Last In First Out) property. It means that the insertion place of the data item is always after the last element of the stack, and the deletion is always clearing the last element of the stack. Hence, the last element in will be the first element out. There is a push() operation which insert the data item as the

last element. There is also a pop() operation which delete the last element. Stacks might be implemented as arrays or linked list [22].

2.6.3 Tree

Most of the following recursive definitions of tree are taken from the Knuth's masterpiece [23]: Formally, the tree data structure is a finite set T of one or more nodes such that

a) there is one specially designated node called the root of the tree, root(T); and

b) the remaining nodes (excluding the root) are partitioned into $m \geq 0$ disjoint set $T_1, ..., T_m$ and each of these sets in turn is a tree. The trees $T_1, ..., T_m$ are called subtrees of the root.

It follows from the definition that every nodes of a tree is the root of some subtree contained in the whole tree. The number of subtrees of a node is called the degree of that node. A node of zero degree is called a terminal node, or a leaf. A non terminal node is called a branch node.

Further, let us define that each root is a parent of the roots of its subtrees, and one child is sibling to another child of the same parent. Note that the root of the entire tree has no parent, and no trees are empty. It has minimum one root node. The level of root of a tree is zero. The level of the root's children is one higher than its parent's level.

The path of a tree is a sequence of node $\{r_1, ..., r_k\}$ where every r_i is the parent of r_{i+1} on that sequence. A full path is a path where r_k is a leaf and r_1 is the root of the whole tree. The length of a path is $k - 1$.

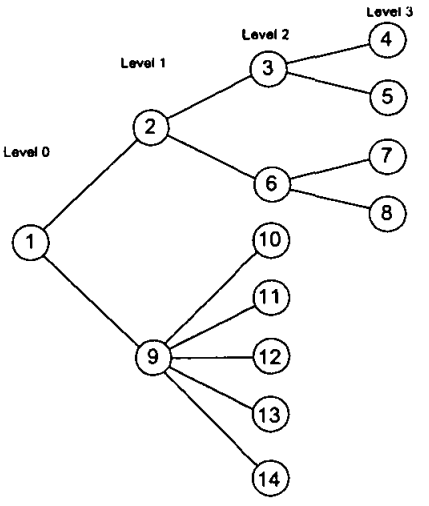


Figure 2.4: An example of tree

In Figure 2.4, the root node is node 1, which has two children: node 2 and 9. Therefore node 2 and 9 are siblings from the same parent node 1. Nodes {1,2,3,4} is a full path, it has the length of 3.

2.6.4 Tree Traversal

Tree traversal means accessing every nodes on a particular tree in a systematic manner. It could be done in many ways, such as Depth First Traversal and Breadth First Traversal. The depth first traversal started from root(T) and proceeds to the first subtree T_1 . In turn, before it process the siblings of that child (root of T_2), the same process is repeated, i.e. another depth first traversal using the subtree T_1. This way, the access is done "depth first" rather than the "breadth" since it reach the highest tree level first. Algorithm 1 is one example of depth first traversal, namely, *preorder* traversal. Back to Figure 2.4, a depth first traversal will access and process the nodes in this sequence: $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14\}$

Algorithm 1 Depth First Traversal: Preorder

```
procedure DepthFirst(T:Tree):
    process the root node of T
    if T is not leaf then
        for each subtree of T, Tc do
            DepthFirst(Tc)
```

The algorithm also illustrates the recursion concept, where it calls itself (to perform exactly the same thing with a smaller scope) somewhere on its parts. Care must be taken to make sure that recursive algorithm eventually halt and returning the result. The responsibility to provide basis for the recurrence lies on the designer of the algorithm.

2.6.5 Backtracking

Backtracking is often used in the context of tree-based solution search, where not all solution space is explored, but instead it stops and backs up and tries different alternatives. Because we can see map-matching as a search of the most resembling route in comparison with the vehicle track, the backtracking concept could also be applied, whereby not all possible solution space is ever explored. Instead it will be based on the vehicle position and its track, and also the possible route alternatives that are still maintained. The term backtracking is also used in the sense of "to change the previous decisions", which will be elaborated more on Chapter 5.1.2.

2.6.6 Measuring the performance of an algorithm

Asymptotic notation is often used to compare the performance (or complexity) of algorithms related to the input size. The Θ-notation asymptotically bounds a function from above and below. When the situation is limited to asymptotic upper bound, the O-notation (called big-oh notation) is used. The following definitions are taken from [24].

The definition for the first notation is given:

$$\begin{aligned}\Theta(g(n)) = \{f(n) &: \text{there exist positive constant } c_1, c_2, \text{ and } n_0 \text{ such that} \\ &0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \\ &\text{for all } n \geq n_0\}\end{aligned} \tag{2.1}$$

The second notation is commonly used, as it is capturing the upper-bound, worst case running time which is common situation in the real world:

$$\begin{aligned}O(g(n)) = \{f(n) &: \text{there exist positive constant } c \text{ and } n_0 \text{ such that} \\ &0 \leq f(n) \leq cg(n) \\ &\text{for all } n \geq n_0\}\end{aligned} \tag{2.2}$$

Asymptotic analysis is done by evaluating the source program or pseudocode and counting the steps of execution especially within loop. These will be summed up in form of O-notation, reducing it to the most significant polynomial terms and omitting the constant coefficients. The reduction is necessary to simplify the analysis, even though it decrease the accuracy of the description. Therefore a tight bounds shall also be sought in order to describe more accurately the performance of the algorithm.

It should be noted that while asymptotic analysis shows the processing time growth in terms of the number of input, it does not reveal the actual processing time. For example, a linear algorithm might perform worse than quadratic algorithm and vice versa, provided that the constant time operation is favouring the quadratic algorithm. But what it guarantees is that at a particular input size and greater, the linear algorithm will outperform the quadratic algorithm.

2.7 Summary

Global Positioning System and other GNSS have several error factors contributing to the inaccuracies in the whole map-matching process. There are many ways to improve GPS accuracy but it entails additional costs which might be too expensive for many situation. Therefore an algorithmic approach might be feasible. Hence, the thesis will not delve

into the GPS technicalities to improve the accuracy, but instead it will use GPS as a blackbox, and assume the standard GPS positioning service, without any differencing or other augmentation such as dead reckoning with the help of other sensor devices (gyrocompass, odometer etc). Consequently, any other sensor support is not required, so it is viable and attractive for worldwide use. The key to the accuracy improvement lies on the ability of the algorithm to perform the process using the standard positioning services and the network topology provided by the digital network representation (road map).

Chapter 3

Literature Review

3.1 Map-matching strategies

This section discusses various geometric-based strategies often found in map-matching. It introduces the naive point-to-point and point-to-curve map-matching, and then proceeds to the more realistic curve-to-curve approach. Artificial Neural Network (ANN) approach is also worth to mention here, although our attempt to use ANN in this thesis work is considered unfruitful. There are some statistical-based strategies, but we will focus more on geometrical strategy.

3.1.1 The point-to-point and point-to-curve map-matching

Simplest point-to-point map-matching algorithm will snap the estimated location P^t to the closest node or shape point in the road network. This notion of "closest" or "nearest" will depend on the measure used, such as Euclidean distance. In the Figure 3.1a this algorithm is illustrated using Euclidean distance as the nearest measure. Therefore it matched three GPS points into three nodes by the translation vectors m1, m2, and m3.

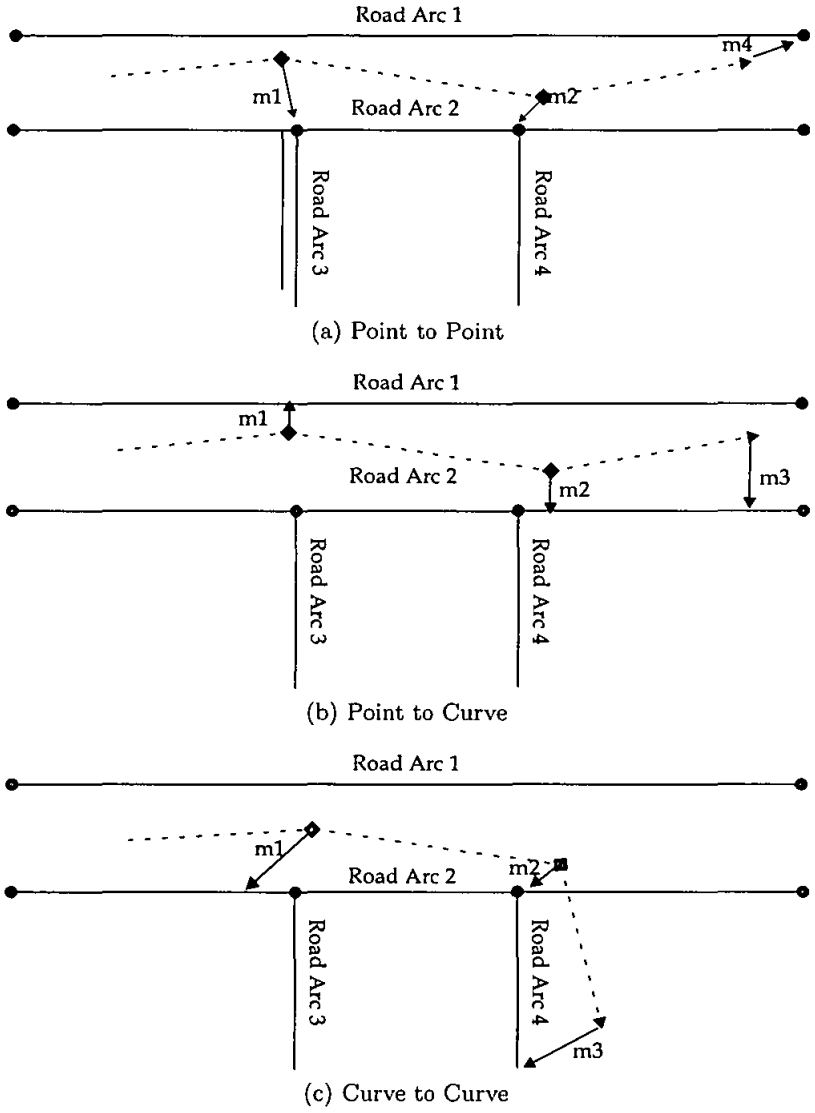
This technique is fast and easy to implement but suffers from too much dependency to the resolution of the arc. If there are two similar roads in parallel, and P^t lies in between those roads, then a road arc with more shape points are more likely to be matched to [3].

The point-to-curve algorithm will select the piecewise arc (or curve) nearest to the estimated point P^t, and find the projection of the estimated point on the curve. If such projection not exists, then it is replaced by the nearest end-point. This is more natural, but may still have the "unstable" property [25], just like the previous algorithm. On the Figure 3.1b, the first GPS point is matched to arc 1. The subsequent points are matched to arc 2. These matches are not contiguous since there are no direct connectivity from arc

1 to arc 2. This is because these algorithms do not care for the continuity and connectivity of the route. A snapped point may be at one time on arc A and suddenly jumps to arc B, then being snapped again on arc A, according to the "nearest" measurement discussed above.

3.1.2 The curve-to-curve matching

A better, albeit more complex, method is to compare the similarity between two curves: the algorithm considers a sequence of estimated positions and matches this to the most resembling arc. Assuming there are m positions, the algorithm is to find the arc (or a combination of it, may be partial) most similar to the piecewise linear arc P, defined by estimate points $(P^0, P^1, .., P^m)$. As illustrated in 3.1c, three estimated points are used to match the last position onto arc 4. On the process, the algorithm infers the previous two positions onto arc 2. This approach needs some way to measure the similarity between two curves, usually by employing some kind of distance metric. The development of the new distance metric being used in this work will be discussed in the Chapter 4.



(a) Point to Point

(b) Point to Curve

(c) Curve to Curve

Figure 3.1: Map-matching Strategies

3.1.3 Artificial Neural Network approach

The map-matching problem explained has several similar characteristics with that of in computer vision or character recognition. The system has to be well prepared for the errors, but still get the correct result. The system could get significant improvement by learning the pattern that exists in the different case but essentially the same. This is pattern recognition, which could be approached with Artificial Neural Networks (ANN).

The ANN as an information processing device composed of highly interconnected nodes has the ability to derive meaning from complicated or imprecise data, and can be used to extract patterns or detect trends that are too complex to be noticed by either humans or other computer techniques.

ANN as Geometric Road Classifier

ANN could be used as geometric road classifier which is trained offline, such as the work presented in [26]. The ANN is trained with a large number of GPS data which must cover all of the expected road geometries in application phase. In other words, GPS data were observed on different transport network (road) categories such as straight roads, curved road parts, parallel roads, junctions and roundabouts [26]. These set of data will be used to train the feed-forward network with backpropagation training algorithm. Although the result is claimed promising, not many researchers are working in this direction.

The handwriting recognition is successfully tackled using ANN approach since each letter still has the same semantic, wherever the position is. The offline training is applied to the ANN so that it recognizes the letter based on the extracted features no matter the position on the paper. For example, the letter A placed on top of the page is still deemed equivalent with the letter A placed on the bottom. Unfortunately, this is not the case with map matching.

In map-matching the existence of two or more similar roads is prevalent. The similarity is often striking, e.g. two parallel roads within some area might be exactly equivalent (their shape is equal). The only differentiating aspect is the position or coordinates of each road. The problem with this approach might be that ANN by itself cannot be used to differentiate two (or more) very similar route hypotheses, if the training is done offline.

Alternatively, ANN could be used as a road geometry classifier which is trained online, and will categorize the vehicle track into one of the possible hypotheses that will come up. The input of the classifier is the important features, such as 2-dimensional simplified pixelation of road network. The output classes are taken from the possible road geometries ahead. The training then done in a real-time fashion, by mapping the inputs into the outputs using typical backpropagation.

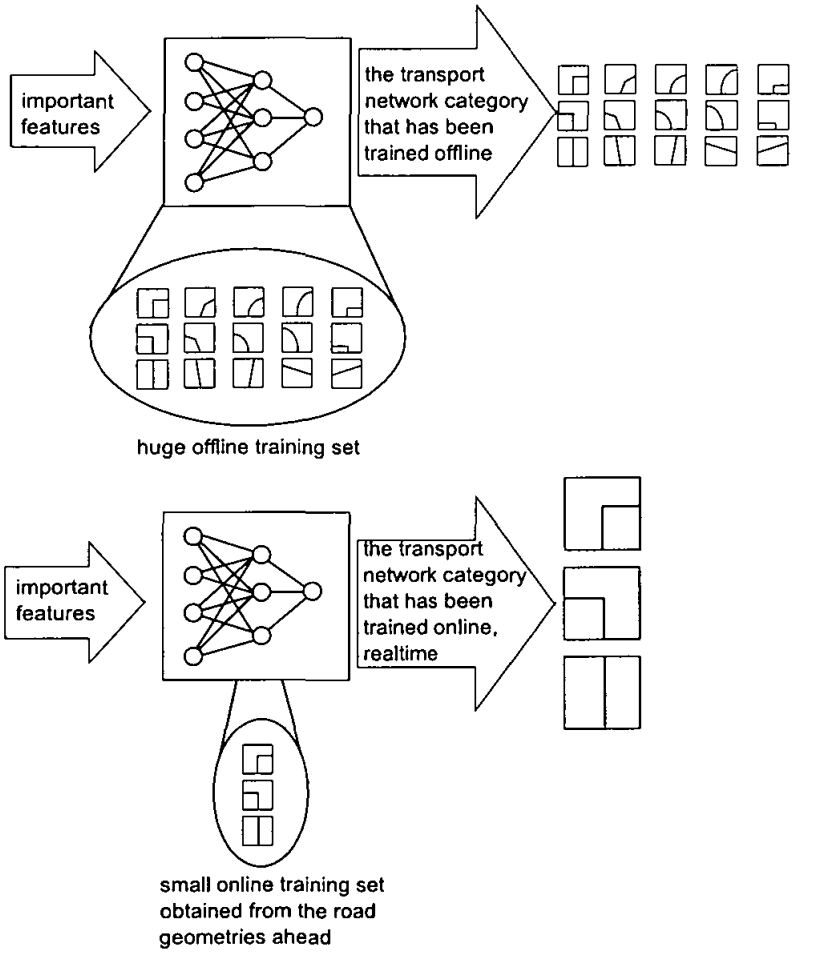


Figure 3.2: Some ANN Solution Architecture

In this online approach, ANN is performing a curve-to-curve matching by classifying the vehicle track into the one most-likely road. ANN exhibits a somewhat error-tolerant characteristic, and is able to recognize patterns which are acquired from the training data. These traits push for a broad range of ANN usage, with backpropagation as the dominant training algorithm, so the application of ANN on the map-matching problem looks attractive.

However, training an Artificial Neural Network with the popular backpropagation typically takes enormous time and computing resource. Hundreds thousands of training iteration might be needed to converge to the correct set of synapses' weights. Unlike online training, the offline training approach will tolerate a lengthy training process.

Unfortunately the sheer size of data input that must be trained makes the offline training approach a bit unattractive.

Not to mention the 'stuck' to the local minima possibility and diverging result which will need some manual intervention (design, tweaking etc). With these inherent characteristics, the conclusion is that backpropagation training cannot be used in real-time fashion. There is an opportunity of using another architecture or training algorithm though, such as Kohonen's Self Organizing Memory(SOM) or Linear Vector Quantization (LVQ) which offer fast-learning [27], but unfortunately these techniques has not been explored enough to warrant a promising result for map-matching.

Both of the above methods is illustrated in Figure3.1.3. There is another way to utilize ANN within map-matching framework, by constructing a predictor for a time series of GPS error. This method will predict the next GPS error based on last N GPS reported location. This idea might be valuable but is not yet thoroughly explored on this thesis. To sum up, the use of ANN is ruled out of this thesis work because of the real-time constraint that is posed by the problem. Instead, a new distance metric will be introduced to perform curve-to-curve matching.

3.1.4 Adaptive-Network Fuzzy Inference System Approach (ANFIS)

An interesting use of ANFIS is explained in [28]. The input of the network are several fuzzy variables routed into several rules producing "the resemblance" as the output. The "resemblance" actually has the same role with "similarity", which determine the similarity of a particular segment with the navigation solution, only that this metric uses non-geometrical approach.

There are several rules incorporated, for example, "if the heading change is nominal and a particular link belongs to a close link set for a larger number of epoch, and the magnitude of velocity is high and the velocity direction is the same as the road link orientation, then the resemblance Z, of that link is high". Of course by adding (or removing) the rules will make the system more or less adapted to the map-matching requirement.

3.2 Global and Incremental map-matching

One could also classify the map-matching algorithm based on its application, i.e. whether the map-matching algorithm has all the trajectory data it needed before it could produce the result (global) or it must cope with local and partial data that is the character of incremental algorithm. It is clear from the real-time usage constraints that we should focus on incremental algorithm. Global (typically offline) map-matching has more flexibility in terms of full points information which tends to make it more accurate. The work in [29] compared the implementation of a global and incremental map-matching and arrived at this conclusion. It is interesting, however, to try to augment the incremental algorithms in such a way that it has the accuracy comparable to that of global map-matching.

3.3 Metrics used in curve-to-curve map-matching

Some kind of distance metric usually is being used to measure the similarity between two curves. Two curves are more similar if the distance between them is lower and vice versa. There are a few properties of a typical distance metric, which calculate the distance

$\delta(x, y)$ between two set of points $x = (x_1, ..., x_n)$ and $y = (y_1, ..., y_n)$, as adapted from [30]:

$$\delta(x, y) > 0 \text{ (non-negativity)} \tag{3.1}$$

$$\delta(x, y) = 0 \text{ if and only if } x = y \tag{3.2}$$

$$\delta(x, y) = \delta(y, x)\text{(symmetry)} \tag{3.3}$$

$$\delta(x, z) \leq \delta(x, y) + \delta(y, z)\text{(triangle inequality)} \tag{3.4}$$

Thus these properties will be important as the constraints and guidelines in the development of the distance metric here. Although the properties are defined for n-dimensional Euclidean space, but the notion of curve could be abstracted from the sequence of points x or y, which makes the above property could still be used for the distance metric of curves.

The Hausdorff distance metric could be used to measure the arc or curve similarity and is easy to calculate, but does not consider the course of the curves. This distance measure ($\delta_H(P, Q)$)is defined below [31]:

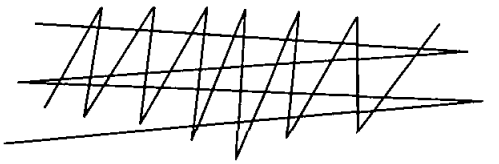
$$\delta_H(P, Q) = \max(\tilde{\delta}_H(P, Q), \tilde{\delta}_H(Q, P)), \text{where}$$
$$\tilde{\delta}_H(P, Q) = \max_{x \in P} \min_{y \in Q} \|x - y\|$$

In other word, the Hausdorff distance from a set of point P to Q could be obtained by first traversing each point $x in P$, and calculating the minimum Euclidean distance from each of x to all points $y \in Q$. This process will give a minimum Euclidean distance for each x. Then the Hausdorff distance from P to Q ($\tilde{\delta}_H(P, Q)$) is the maximum of those minimums. Note that this Hausdorff distance from P to Q ($\tilde{\delta}_H(P, Q)$) may have different value compared to the Hausdorff distance from Q to P ($\tilde{\delta}_H(Q, P)$), therefore the *symmetry* property is violated. In order to avoid asymmetry, the formula is therefore designed to pick the maximum of the two, giving $\delta_H(P, Q) = \delta_H(Q, P)$.

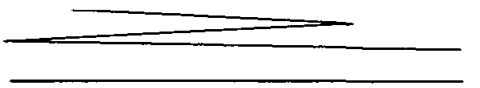
As opposed to Hausdorff distance, Fréchet distance consider the course of the curve and presents a better similarity measurement for the example set in Figure 3.3 reproduced from [32]. This example curves will be used to validate the proposed distance metric later in Chapter 4.

The Fréchet distance could be illustrated by the following popular definition adapted from [33]: A man is walking a dog on a leash: The man can move on one curve, the dog on other. The Fréchet distance is then the minimum length of leash that is sufficient for traversing both curves, provided that both may vary their speed (in order to get the minimum length of leash), but backtracking is not allowed.

The drawback in Fréchet is perhaps on the running time and the complication of



(a) Two curves resulting a small Hausdorff distance and large Fréchet distance



(b) Two curves resulting a small weak Fréchet distance and a large Fréchet distance

Figure 3.3: Fréchet and Hausdorff Distance Comparison

implementing the algorithm (we assume the implementation is using a parametric search technique presented by [34]). It solves the problem in the order of $O(pq\ log^2 pq)$ (where p and q are the number of segments of polygonal curve [33]), and the huge constant operational time might not fit enough for real-time constraint. There are global offline, post real-time algorithms which use a weaker scheme of Fréchet distance, such as [29], in order to speed up the running of the algorithm, but nevertheless it is still slow compared to the incremental map-matching. The approach of [35] is using a Quicksort (instead of parallel merge sort as exemplified in [34]) and achieve a better result. Unfortunately we still consider it too slow for our constraint.

In an attempt to satisfy the near real-time constraint, we decided to implement a faster way to compute distance (or similarity, by the opposite sense) between curves. One way is by using parameterized (unidirectional) curve, as presented in [8]. The distance between two curves A and B, assuming that the curves are parameterized in $a : [0, 1] \rightarrow A$, can be written as $\|A - B\| = \int_0^1 \|a(t) - b(t)\| dt$. This distance metric will be the basis for *arc distance* which will be used as the main map-matching technique on this project.

3.4 The Framework for Map-matching Metrics

It is clear that the distance metric alone could not perform the map-matching, especially in the curve-to-curve map-matching setting. This section will take a survey among "framework" algorithms which will employ a particular distance metric into a good use. We will use the term *hypotheses* loosely to represent many possible road alternatives. The general idea is to find the minimum distance hypothesis, compared to some or all part of the vehicle track.

3.4.1 Road Reduction Filter (RRF)

Taylor et. al. [36] describe a map-matching algorithm which uses height information from the map's digital terrain models to better assist the GPS to improve the accuracy, even when only three GPS satellite is visible. It maintains and monitors some hypotheses until the difference in heading and position is passing some threshold, whereby it would discard that particular hypothesis. The RRF maintains the road hypotheses for a time constant (30 seconds/epochs), then it will provide the estimate correct road based on the hypotheses' correlation with the vehicle track. The distance metric employed here is the difference in bearing and position of the hypotheses (or might be called pseudo-measurements) with the vehicle trajectory. The initial matching process is geometric curve-to-curve matching, which is quite sensitive to outliers [1]. It is interesting to analyze that if for some time constant (say near 30 seconds/epoch) there are no significant vehicle movements, such as found in traffic lights or the vehicle is parking, then there might not any sufficient information available to reduce to a single road correctly.

The original RRF does not need complex data structure to maintain its hypothesis. This simplicity is sufficient since RRF does not use the road connectivity information [1]. The drawback of ignoring connectivity is that jumping, unstable map-matching result might materialize. The road connectivity could also bear important information such as traffic direction, so it might not be optimal to ignore the road connectivity completely. However, newer version of RRF has evolved to incorporate network analysis software to cater the road connectivity, including the driving restriction information [13].

The RRF is claimed to converge the hypotheses to the correct road for only a few seconds in most cases, but has a problem in "along track error". That is, RRF could infer the correct road but it could not determine the spot on that road where the vehicle is located. The development of so-called Mapped Dilution Of Precision (MDOP) is an attempt to solve this problem, which is explained in [37] and [13].

One last note is that RRF does not have the backtracking ability, i.e. to rewind the less likely hypothesis and picks the more likely hypothesis to be presented to user. Once the choice is made, it is never changed. While this provides a bold information to user, the flexibility and accuracy is traded for that. More discussion about backtracking is presented in 5.1.2.

3.4.2 Multiple Hypotheses Technique

A pioneering work of Reid, later popular as Multiple Hypotheses Technique (MHT), spawns many researches and applications. It can be used to track multiple targets in a cluttered environment, by associating measurements with the appropriate tracks of

the targets movement [38]. MHT will generate a set of data-association hypotheses to account for all possible origins of every measurement. In other word, given a set of imperfect measurements of multiple moving targets/objects, MHT could give the estimate description of which object has which track, including the new object which appearing in the middle.

MHT is based on Bayesian probability to give the estimation on measurement-tracks association hypotheses. However, we are inspired in its management of hypotheses, rather than its calculation of the most likely hypothesis using Bayes' theory.

The hypotheses can be shown as a tree (represented using a two dimensional array in computer), which lists measurement-oriented hypotheses, i.e. every possible target is listed for each measurement. A measurement represented as a level of that tree. Each node on a particular level is a possible target for a measurement. This technique is formulated for multiple targets, and a Vehicle Tracking Systems brilliant research by reformulating Multiple Hypotheses Technique to a single target problem can be found in [19].

The use of tree data structure enables the system to store a huge number of hypotheses in an efficient storage with minimum redundancy. Also, the huge hypotheses will be useful to a very flexible backtracking, not limited to some time constant. By managing the hypotheses as a tree we can employ the distance metric that is appropriate to evaluate those hypotheses. In other words, the tree data structure is very important part of the frameworks algorithm that will be developed.

3.5 Measuring Map-matching accuracy

A brief exposure on the map-matching accuracy is already given in Chapter 2.1. We will proceed with the discussion in the map-matching context, starting with the work of Morisue [39] for the indices 1-3 below, and adding them with the index used by [3] (the fourth index).

The map-matching evaluation indices can be one of the following: 1. Average mileage driven until the vehicle is off (map-matching limit).
This index records the travel length until a map-matching technique fail to match the map with the vehicle track. It is quite normal that, eventually, the map-matching algorithm fails to produce sensible match because wrongly selected earlier route or other deficiencies in the process.

2. Average location accuracy.
This index is determined from the average of error of the matched route as compared to the "true" route. The true route might be obtained from the differencing techniques

such as RTK or DGPS. Alternatively, as a more coarse approximation, the index can be determined from the error average of the matched points as compared to a list of specified known benchmark points location.

3. Wrong route driving ration.

This is the ratio of driving distances on wrong routes, divided by distances driven on correct routes. The index is useful because wrong route indication often confuses driver, especially in the urban area settings where there are too many roads intersects one another.

4. Correct arc ratio.

This index results from the ratio of the number of correctly matched points divided by the total number of matchings. A match is correct if the vehicle is on the same arc predicted by the system.

We will use the fourth index as it is relatively easy to calculate without sacrificing the "perceived" accuracy of the human user. The first and second index might be giving high score for a system which often fail to give the correct route. In other situation, a good map-matching algorithm which provide a correct route on a largely distorted map might be considered low in accuracy according to the first and second indices.

3.6 Summary

There are a number of map-matching methods to improve the accuracy. Geometrical methods works by examining the geometric properties of the road arc and the vehicle trajectory. Statistical approach involves probabilistic estimation of the correct road link given set of measurements and history of vehicle motion.

Our interest is on the geometric curve-to-curve matching. This method compares the curve of vehicle track and the set of curves of possible road routes (measured in a particular metric) and presents the estimate route based on the most similar curves combination. There are several metrics that can be used in the curve-to-curve matching, including Hausdorff and Fréchet distance metrics.

A map-matching algorithm could also be seen either as global (offline) or incremental (online). The first has all the information upfront while the latter obtain the information partially through time. The global map-matching is more accurate than incremental map-matching but not suitable if used in a real-time situation. The challenge is to have an incremental map-matching which is comparable to the global map-matching in terms of accuracy.

The accuracy itself could be measured using several indices. Of those indices one is picked to evaluate the performance of the proposed algorithm. The choice is based on

the user-relevant property and it is also relatively easy to calculate.

Chapter 4

The Development of *Arc Distance Metric*

As noted in Chapter 3, curve-to-curve matching will need a measure on curve similarity. On this chapter, a new metric will be developed, that will be referred as *arc distance*. The arc term is used instead of curve, as typically used in the GIS environment where the vehicle tracking systems will be applied. An arc could be viewed as piecewise linear curve, which typically would represents an actual road in the road network system.

The most important criterion of the design is that the metric should measure similarity between curves (or arc) visually, by tolerating some translational error. This is because we will use the metric for measuring arc similarity in the context of map-matching. Further reasoning about this is described in Chapter 5.1.1.

4.1 The Metric Design

The basic curve distance metric (version 1) to measure similarity of two curves A and B which will be modified is reproduced here for easier reference:

$$\|A - B\| = \int_0^1 \|a(t) - b(t)\| dt \tag{4.1}$$

It should be kept in mind that for two curves to be more similar, the distance metric should give lower value, and for two identical curves, the distance metric should return zero.

Note that we use capital letter (A) to represent the curve as a whole, while the lowercase ($a(t)$) is used to represent the parametrized version of the former (A). This style will be used throughout this discussion.

The Figure 4.1 explains the computation procedure of this metric. The first step is

to set two "cursors" for each of the curve, then the cursors are moved toward the end of the curve for some distance. The cursor is simply a variable to record a two-dimensional position. For a lengthier curve, the distance step is bigger than the shorter curve. For same-length curve, the distance step will be equivalent. The distance step depends on the length of each curve. At each step the Euclidean distance is calculated, then added to the total sums. The distance between the curves is then the total sum multiplied by stepping parameter dt.

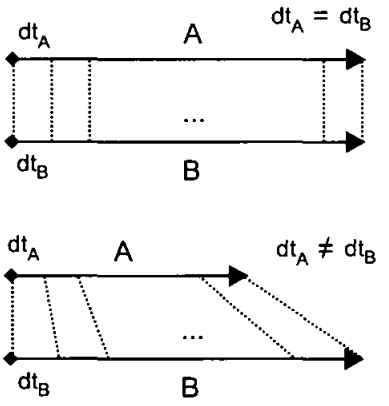


Figure 4.1: Metric version 1 computation

The problem with this measure is that it will not operate very well on curves with quite different length [36]. Given two curves in differing length, we would like to measure more on the initial similarity (that is the initial subcurve of the longer curve), rather than measuring the whole curve similarity. This is because –in map matching– the curves constructed by the vehicle track, or the road track, could be considered in the same scale or unit. That is, the accuracy problem is mostly on a translational error and not on the scaling or rotational error.

We will modify this so that it may handle the curves with different size good enough. In other words, we want to measure more on initial similarity and not the whole curve similarity. This is done by picking the shorter curve of the two as $Q = min(A, B)$, and the longer $R = max(A, B)$, and R_{init} denotes the initial portion curve of R with the same length of Q, then

$$\|A - B\| = \int_0^1 \|q(t) - r_{init}(t)\| dt + D_r \tag{4.2}$$

where D_r is the residual distance, which is

$$D_r = \int_0^1 \|q(1) - r_{res}(t)\| \tag{4.3}$$

where R_{res} is the last portion of R which follows the equation $R = R_{init} + R_{res}$. This way, this new metric (version 1) will emphasize more on the initial portion of the curves rather than the full curve comparison of two differing sized curves.

This modification is not complete yet. We found that the metric still has a problem of accounting too much on the distance between the points within the curve (inter-curve distance) rather than the shape of the curve itself. The illustration is presented on Figure 4.2. On the figure, the curve P and Q is exactly the same in terms of shape,

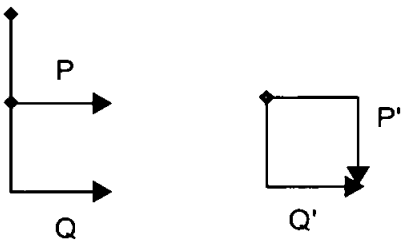


Figure 4.2: Equivalent curves is considered dissimilar compared to a very different curves

but they are separated by some (inter-curve) distance. The curves P' and Q' is totally different but positioned nearby. The modified equation will give a very similar result, which is not suitable for the map-matching case. In map-matching , some slight, quasi-translation is quite common, as can be seen in the Figure 5.1.

Therefore, the shape of the curve should be accounted much rather than the distance between two curves. To make the metric account the shape of the curve, it is necessary to compare both curves from the same origin point. This idea lead to the second modification.

The second modification (algorithm version 2) is done by translating one of the curves so that the initial points of those curve coincides. In other words, two curves will have the same origin. Let R' is the translated curve of R, such that (note that $r(t)$ is a parametric function of curve R)

$$r'(t) = r(t) - T \tag{4.4}$$

where T is the translation delta, obtained from the initial points difference.

$$T = r(0) - q(0) \tag{4.5}$$

This R' is to replace R in our previous metric. The motivation behind this aligning is to focus on the curvature of the curves, and not the Euclidean distance between the curves. As illustrated in Figure 4.2, two curves that are very dissimilar could have the same similarity as two identical curves, provided that some special arrangements on the

curve position are made.

We may note that the last modification will discard the Euclidean distance factor, i.e. two curves have the same similarity, even if we separate those curves away to a very far distance. This is not suitable for map-matching which prefer closer road arcs as the better matching hypothesis, provided both have identical curvature. So the last modification (version 3) is to account the maximum distance between two curves M, as a multiplier of the metric. This variable is calculated from the larger value between M_{init} and M_{res}, i.e. $M = max(M_{init}, M_{res})$, where:

$$M_{init} = \max_{t\in[0,1]} \|q(t) - r'_{init}(t)\|$$
$$M_{res} = \max_{t\in[0,1]} \|q(1) - r'_{res}(t)\|$$

Note that M_{init} is the maximum distance between the shorter curve to the initial portion of the longer curve. Similarly M_{res} is the maximum distance from the end point of the shorter curve Q to the last portion of the longer curve R. Thus the final modification should read as:

$$\|A - B\| = M \int_0^1 \|q(t) - r'_{init}(t)\|dt + D_{r'} \tag{4.6}$$

This distance metric will be further referred as *arc distance metric*, represented by the symbol δ_{adm}, and will be used as the primary method on matching the vehicle tracks to the correct road arcs.

4.2 The Implementation

The metric is implemented in Algorithm 2 below, accepting two curves in the form of polylines. A polyline is considered as an array of coordinates. The Trace() procedure is responsible for tracing the two polylines starting from some point for some length specified (in the algorithm it is represented as delta length dL). Tracing is the process of moving a cursor in the curve direction for some length. The important data is the new cursor position on the curve after trace, upon which the distance between two cursors is measured. The distance will be multiplied with dL, and will be topped up to the total distance. The distance metric is then the total distance after both lines are traced. The algorithm should be more clearly explained by Figure 4.3.

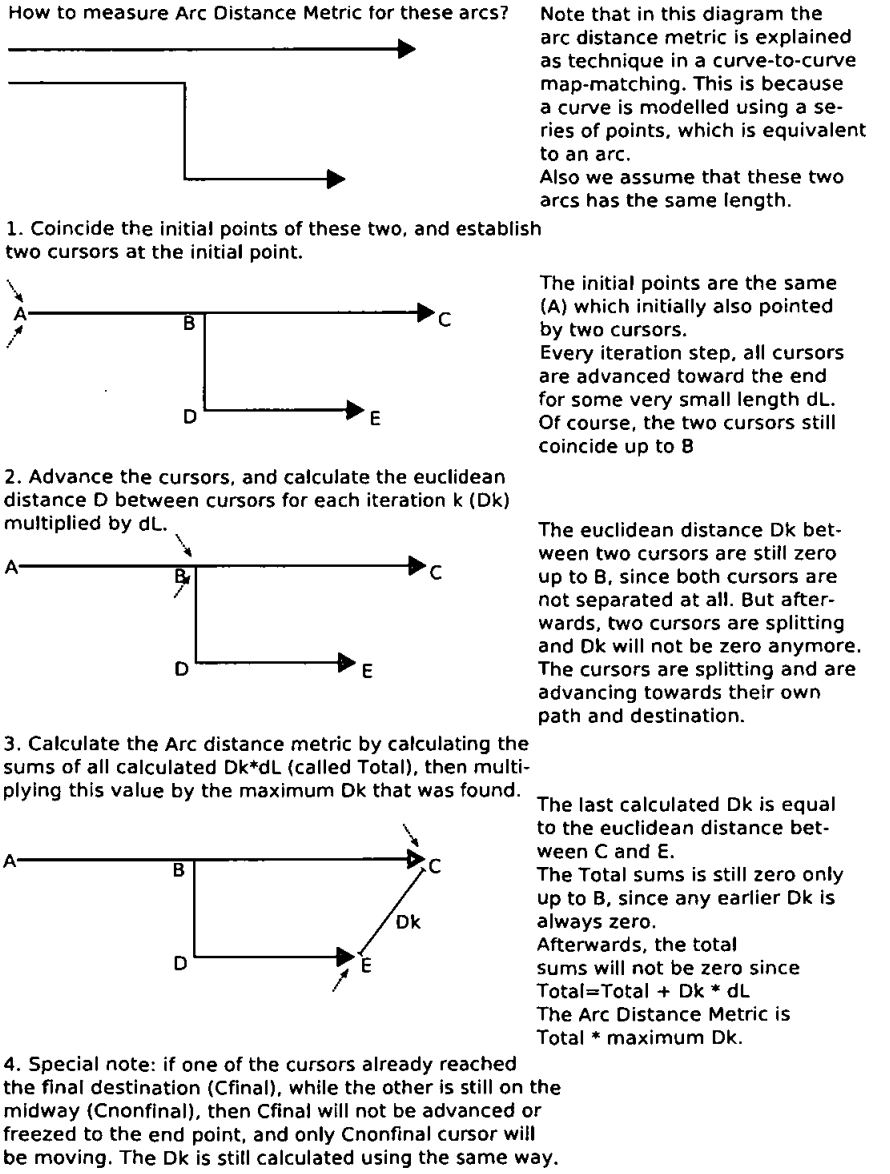


Figure 4.3: Illustrated explanation to calculate the *Arc Distance Metric*

4.3 The *Arc Distance Metric* in the Light of a Distance Metric Criteria

Section 3.3 discuss about general distance requirements, upon which *Arc Distance Metric* will be scrutinized. The observation of *Arc Distance Metric* algorithm shows that the metric at least has the following properties (please refer to the figure 2 for the variables used here):

4.3.1 Non-negativity ($\delta_{adm}(x, y) > 0$)

Since D is derived from Euclidean distance and always multiplied by a positive constant, then consequently the Total (which is started from zero) is always non-negative.

4.3.2 The distance is zero for equivalent curves ($\delta_{adm}(x, y) = 0$ if and only if $x = y$)

For equivalent curves, the Euclidean distance per segment D is always 0. Therefore the Total is always 0 for this case. For non-equivalent curves, there must be a different cursor location at particular step somewhere. Thus the D will not be 0, and of course the *Arc Distance Metric* result will not be 0. Note that there is no cancellation effect that will affect the final result since D is always positive.

4.3.3 Symmetry ($\delta_{adm}(x, y) = \delta_{adm}(y, x)$)

The Euclidean distance D itself is symmetric, therefore it will not change if the polylines' order in the metric's parameter is reversed. Now the Polyline3 is arbitrarily derived from Polyline2, and not Polyline1. Even though if it is derived from Polyline1, D will not change because of the simple translation. The experiment with the algorithm also shows this result.

4.3.4 Triangle Inequality

For this fourth property (triangle inequality) it might take a further mathematical proof and will not be discussed here. Basically the *Arc Distance Metric* works and these three properties are quite satisfying for the application in map-matching process.

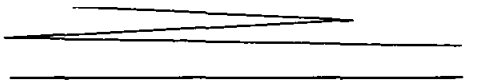
Algorithm 2 Outline of Arc Distance Metric

```
function ADM(PolyLine1,PolyLine2):

    translate Polyline2 so that its initial point = Polyline1's initial point
    Pos1 = initial point on PolyLine1
    Pos2 = initial point on PolyLine2
    Total = 0.0
    repeat
        Pos1 = Trace(PolyLine1,Pos1,dL)
        Pos2 = Trace(PolyLine2,Pos2,dL)
        D = Euclidean distance between (pos1,pos2)
        Total = Total + D * dL
    until all lines are Fully Traced
    return ( Total * (maximum D found during the above loop) )

Trace(PolyLine,StartingPosition,L):
    if StartingPosition is at the end of the polyline then
        signal Fully Traced for this line
    else
        move the StartingPosition cursor toward the end of polyline for L distance
```

4.4 Simple benchmarking

The metric is used to compare these two sets of curves in Figure 4.4. The lesser the distance, the more similar they are. A good distance metric should result a zero distance if applied to two equivalent curves, and also return large distance to both of these sets. These sets are also used to benchmark between the Fréchet and Hausdorff distance in Chapter 3.3, so it is also used here to test whether the *arc distance metric* (ADM) might perform as good as Hausdorff distance metric.

(a) Set 1: Small Hausdorff distance, Large Fréchet distance, Large ADM distance



(b) Set 2: Small weak Fréchet distance, Large Fréchet distance, Large ADM distance

Figure 4.4: ADM, Fréchet and Hausdorff Distance Comparison

Both figures are represented in pixel coordinates. For Figure 4.4a there are two curves, A (having many zigzagged lines) and B (less zigzagged lines), which have the coordinates as listed in Table 4.1 (for curve A) and in Table 4.2 (for curve B).

Point#	X	Y
0	57	134
1	118	22
2	99	145
3	175	22
4	161	153
5	234	22
6	218	171
7	276	24
8	266	192
9	335	28
10	323	169
11	398	32
12	397	163
13	485	40
Length	1912,33	px

Table 4.1: Set 1: Curve A points

Point#	X	Y
0	46	40
1	515	72
2	26	106
3	531	127
4	15	174
Length	1983,84	px

Table 4.2: Set 1: Curve B points

For this pair, ADM returns $\delta_{adm} = 159480758,678852$. This is a large value indicating dissimilarity of the pair. Now for the set displayed in Figure 4.4b there are also two curves, A and B, which is having the coordinates listed Table 4.3 (for curve A) and in Table 4.4.

For this second pair, ADM returns $\delta_{adm} = 130648470,529814$. This is also a large value indicating dissimilarity of the second pair. We will compare this result with the Hausdorff distance and Fréchet distance soon.

The conclusion that might be drawn from the sets is that the *Arc Distance Metric* is better for measuring similarity of two curves or arcs, at least if compared with Hausdorff

Point#	X	Y
0	78	115
1	405	131
2	25	150
3	520	159
Length	1202,95	px

Table 4.3: Set 2: Curve A points

Point#	X	Y
0	32	195
1	521	195
Length	489	px

Table 4.4: Set 2: Curve B points

distance. Also, the performance is comparable to that of Fréchet distance. We will put this metric into real test on the field experiment explained in Chapter 7.

Chapter 5

Managing multiple hypotheses

The development of the arc distance metric on the Chapter 4 provides one basic needs for curve-to-curve map-matching to measure the curve-to-curve similarity. The next step is to employ that distance metric into a larger algorithm which will be responsible to decide what and which arc to compare with the vehicle track (the hypotheses generation) and how to find the most likely arc (the hypothesis selection) which will be presented to user as the route for the vehicle. In other words, a suitable framework map-matching algorithm is needed to manage the multiple hypotheses, which will utilize the *arc distance metric* in an optimal way. The survey about the available framework algorithms is presented in Chapter 3.4.

5.1 Algorithm Design

5.1.1 Spatial Mismatch

The diagram in Figure 5.1 illustrates a typical scenario in map-matching problem. To simplify the explanation, the illustration assumes that the road map is perfectly accurate. The vehicle was moving along Street 1, then turned right onto the Street 2, and turned left onto Street 5. This actual track is shown by the solid arrow inside the road. The estimate positions given by the navigation system (GPS or any satellite-based) is shown by the dotted arrow which is resembling (or similar to) the actual track, but slightly translated to the northeast direction. This is because the estimate positions from GPS are often having a slow varying error, so for, say, 60 seconds scale it is quite possible to have situation like depicted on the diagram. The *spatial mismatch* will happen by the time the system presents erroneous inference such as snapping to $\vec{P}^a$ for the GPS point at a.

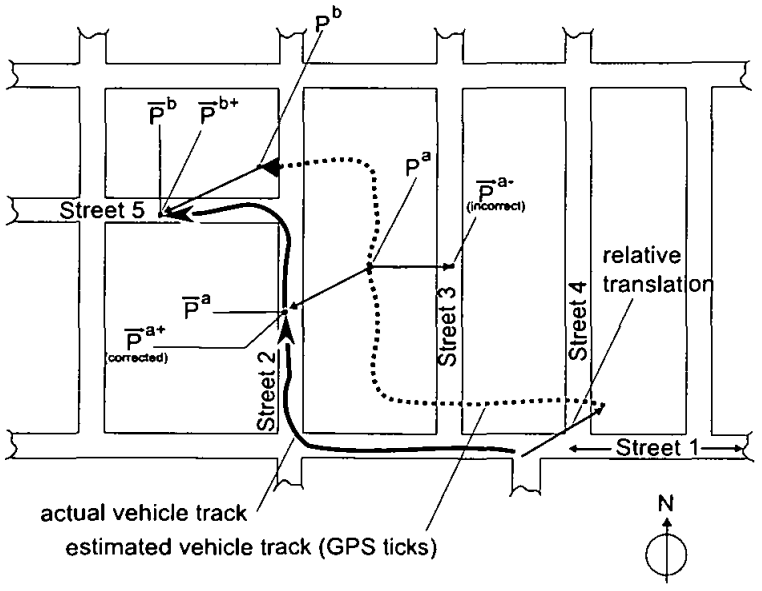


Figure 5.1: A scenario where backtracking is necessary

5.1.2 Incremental Algorithm with Backtracking Ability

On the previous Figure 5.1, the navigation system reports P^a as the position of the vehicle at time a. Assume that the real vehicle location is $\overline{P}^a$. Later at time b, a similar observation could be made, returning P^b as an estimate to $\overline{P}^b$.

Using such assumptions, an incremental map-matching algorithm has to provide best estimate of the route and position at any time a and b (that is, P^a and P^b) which coincides on the road network. It could be seen that at time b, there is enough information to say that –by visual inspection of the estimate track and the road network– it is likely that the vehicle took the direction from Street 1, then turning to Street 2, then again turning to Street 5. It is supported by the fact that Street 3 alternative is improbable, since it has no left turn until some distance, quite far ahead. Whereas back at time a, such information is not yet available, forcing immature inferences from a set of hypothesis, i.e. either: (1) The vehicle took Street 1 then turned to Street 2, or (2) The vehicle took Street 1 then turned to Street 3. Let us say that the system came up with wrong inference and picked $\overrightarrow{P}^{a-}$ for that time (i.e. the option number 2). The negative sign symbol is used to hint the reader that this will be an erroneous choice. The system does not know nor use that symbol. At time b, the system decided that it needs to backtrack and modifies the previous choice $\overrightarrow{P}^{a-}$ into $\overrightarrow{P}^{a+}$, since that is the most likely road taken by the vehicle.

This scenario illustrated that the partiality and incremental nature of the problem needs to be considered carefully to come up with a good map-matching algorithm. In other words, a good incremental map-matching algorithm should be able to *backtrack* to the most likely hypothesis available.

The backtracking should not to be implemented as a reduction of hypotheses, such as reduction within particular time limit like found in Road Reduction Filter (which maintains hypotheses only for the last 30 epochs/seconds) but instead, as long as the hypothesis is good enough (that is, quite resembling to the vehicle track), then that hypothesis should be kept and not discarded. Therefore it will provide the ability for the system to backtrack and amend the previous choices whenever necessary.

In order to do a decent backtracking, the system needs to manage the hypotheses in an efficient manner, by storing the necessary data into a suitable data structure for the purpose. It is why the Multiple Hypotheses Technique offers an interesting insight to the problem since it models the hypothesis in an efficient tree model. This thesis took the similar direction of Pyo, by reformulating the Reid's Multiple Hypotheses Technique as a single target problem, and generating pseudo measurement using adjacent road networks [19]. This thesis do not, however, follow the Multiple Hypotheses Technique use of Bayesian probability to do the comparison, but will instead use the *arc distance metric* developed in Chapter 4.

5.1.3 The Algorithm Description

We will start with the hypothesis definition. Each hypothesis H inside $\mathcal{H}$ consist of sequence of points $(H^0, H^1, ..., H^{nH})$ and a corresponding arc segment for each H^n. It means that each hypothesis is assumed as a possible vehicle route through the road-center lines, starting from the spot referred by the first hypothesis (H^0) and proceeds through the sequence of hypotheses until the last spot (H^{nH}). It also uses the tree representation, consisting of a root node and a number of hypotheses tree H as its children. A hypothesis is stored as a full path of that tree, from root to the leaves, where H^n is a direct parent of H^{n+1}. It is easy to see that the new hypotheses "grow" on the previous hypotheses, by noting that new children will inherit the same ancestors in that tree. In other words, new hypotheses are generated by expanding a leaf node on the hypotheses tree.

The hypotheses tree is illustrated in Figure 5.2b. As already mentioned in the previous section, the map-matching process does not use any probability calculation of each hypothesis. Instead, the *arc distance metric* measure will be used to evaluate each of the hypotheses to pick the best road the vehicle is on.

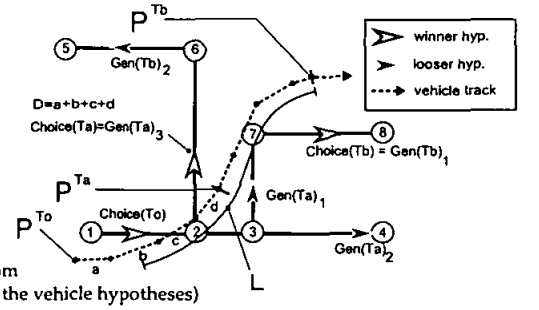
The central idea of the solution is to look back and ask two things:
1) what is the shape of curve constructed by the vehicle track just now, and
2) what is the possible route that is most similar to that shape of curve.

Therefore the algorithm has to evaluate the curve similarity between last estimated vehicle track and each possible route according to the map. The objective is to find the best estimate route (hence the road as well), which is most similar to the vehicle track (i.e has the lowest *Arc Distance Metric* property). The route can be composed of many connected roads or road segments, as long as the total length of the route is equal to the last travel length of the vehicle for some period.

In the Figure 5.2a, the real route taken by the vehicle is through the node $(1, 2, 3, 7, 8)$. The bold dotted arrow at the center represents sequence of GPS-reported estimated positions (i.e. the estimated vehicle track). The corresponding hypotheses tree for the scenario can be seen in Figure 5.2b. For the explanation purposes, the hypothesis point H^n information will be omitted, and the discussion would use the road segment information instead. Therefore a hypothesis will be represented as sequence of node such as $(1, 2, 3, 7, 8)$, which has the meaning that the vehicle went through nodes 1,2,3,7 and is currently lies somewhere between node 7 and node 8. This simplification will be used to convey the conceptual discussion. For a more detailed hypotheses tree structure, the reader is suggested to refer to Section 5.1.5.

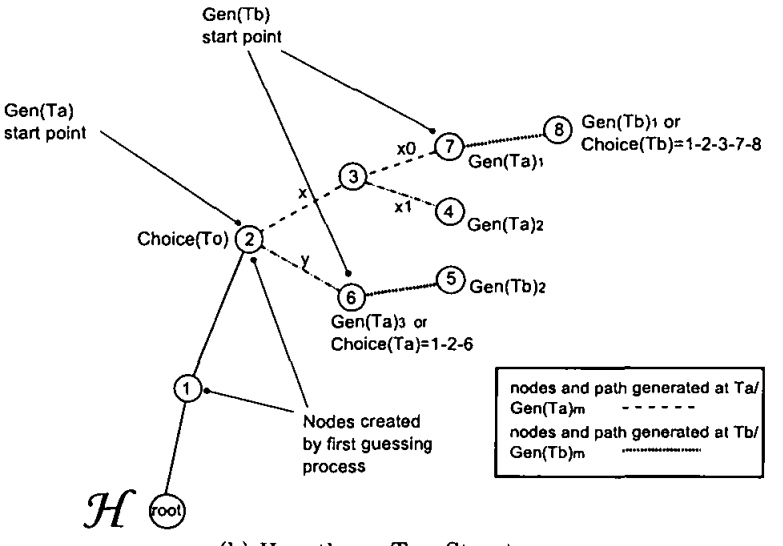
1 The system generated the hypotheses at Ta, Gen(Ta). Using arc similarity, it picked the most probable hypotheses at that time, Choice(Ta). This time it (temporarily) incorrectly infers the vehicle route as (1-2-6).

2 At Tb, the system will produce Gen(Tb), by growing the previous hypotheses set by the growth length D (the estimated last travelled distance calculated from the last update is used to "advance" the vehicle hypotheses)



3 Among the new hypotheses is that (1,2,3,7,8) and (1,2,6,5). The hypotheses (1,2,6,5), represented by Gen(Tb)2 is the result of growing Gen(Ta)1. Notice that Gen(Tb)2 is exactly D distance grown (following the road) from Gen(Ta)3. All hypotheses will be compared with last travelled vehicle track L for the similarity. It is clear at Tb, that the more probable route is (1,2,3,7,8) since there is a sharp right turn which is only possible on that route. Indeed, the system use arc distance metric and picks Gen(Tb)1 as the winner for epoch Tb/Choice(Tb).

(a) The Map and Vehicle Track



(b) Hypotheses Tree Structure

Figure 5.2: Hypotheses Tree Generation and Choosing the Winner

Let $(T_0, T_1, ..., T_\infty)$ indicates a sequence of timestamp within a fixed time interval (e.g. 5 seconds), each comprising of several P^t information (for example, if the GPS epoch/ticks is set per second then each five seconds interval would comprised of five estimate points P^t). Each T_n is called an "update", and it holds that T_j happens earlier than T_k if $j < k$. Each update is the time for the system to update the hypotheses and do map-matching based on the retrieved information from navigation system. The figures shows that at T_a, the system must find on what road the vehicle is on, based on the previous vehicle track in form of estimate positions up to $\mathcal{P}(T_a) = (P^{T_0}, P^{T_1}, P^{T...}, P^{T_a})$ and also the previous hypotheses set. In the illustration, it is $\mathcal{H}(T_a) = \{(1,2)\}$ (to simplify the illustration, assume that is the only hypothesis at T_a). This is where the curve-to-curve algorithm performs. It must find all possible, adjacent curves (roads) from all previous hypotheses, to compare it with the last vehicle track for some distance C. In order to do that, it must grow the hypotheses tree $\mathcal{H}$ for some growth-distance D, which is estimated from $\|P^{T_a} - P^{T_{a-1}}\|$. That is, D represents the distance traveled by the vehicle since last update. Usually the setting is that $C > D$. The algorithm then picks the most likely hypothesis (the one with minimum sums of all *Arc Distance Metric* performed for that hypothesis) tentatively and the system will present that as the estimate vehicle position on the road and also as the estimate vehicle route. Back to the figure, at T_a, there are at least three new hypotheses: the vehicle went through the route (1,2,6), or the road (1,2,3,7), or (1,2,3,4). The hypothesis (1,2) is not included because it is the old hypotheses upon which the new hypotheses must grow. Let say that the *Arc Distance Metric* score the minimum for the sequence (1,2,6) and makes it the winner for this update, but because (1,2,3,7) and (1,2,3,4) is quite "good", then these two hypotheses will remain for some time.

To delve more detail on the process, the content of the hypotheses tree is displayed in Figure 5.2b, and the step-by-step execution is illustrated in Figure 5.3. Before T_a, there is only one hypothesis available in the tree, $\mathcal{H} = \{(1,2)\}$. This is the result of the first guess performed at T_0 using simple point-to-curve matching, since there are no sufficient information available (i.e. no hypotheses yet) at the first time. Note that the nodes 3,7,4, and 6 is generated on the grow phase of T_a (noted by $Gen(T_a)$). After the growth of the tree (hypotheses generation) of T_a, $\mathcal{H} = \{(1,2,3,7),(1,2,3,4),(1,2,6)\}$. This generated hypotheses $Gen(T_a)$ is drawn using dotted line on the Figure 5.2b and is "grown" over the previous hypotheses $\mathcal{H} = \{(1,2)\}$.

The next update, T_b, the system will again grow the tree (the generation process noted by $Gen(T_b)$), this time the growth distance D can be seen in Figure 5.2a, step 3. This will grow (i.e. branch) two leaves to the hypotheses tree that is the two new hypotheses: a) (1,2,3,7) added with the node 8, becomes (1,2,3,7,8) and

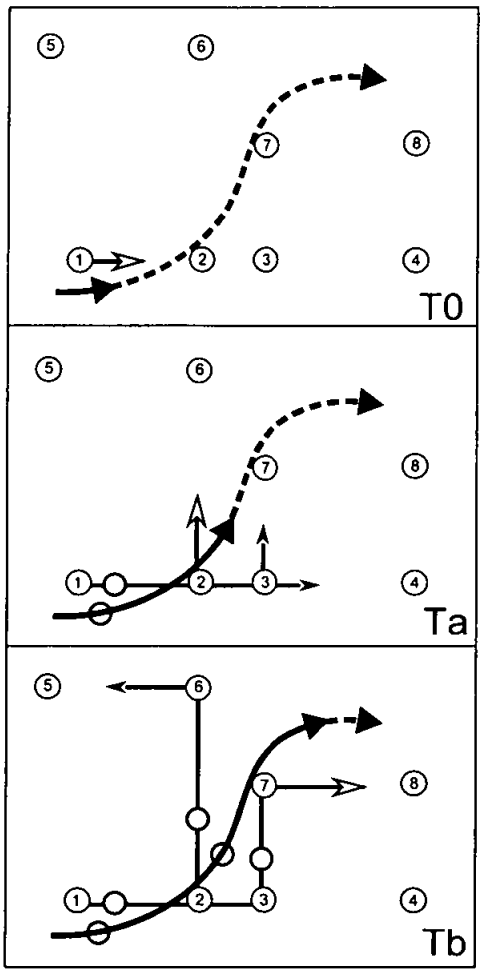


Figure 5.3: Hypotheses Generation Step By Step

b) the addition of node 5 to (1,2,6), which will become (1,2,6,5).

Notice that there are no growth on the (1,2,3,4) branch because it is too dissimilar to the last vehicle track. Whenever the *Arc Distance Metric* of a branch yield $D > D_{max}$, it will be discarded immediately and will not be added to the hypotheses tree. Also, it is clear now that (1,2,3,7,8) is much more similar compared to (1,2,6,5). This is backed up by the evaluation of *Arc Distance Metric* for those hypotheses. Note that the winner-decision process will consider the sums of all previous *Arc Distance Metric* that has been done for a particular hypothesis. It means that $\delta_{adm}(1,2,6) + \delta_{adm}(1,2,6,5) > \delta_{adm}(1,2,3,7) + \delta_{adm}(1,2,3,7,8)$. In fact, at T_b, the hypothesis (1,2,3,7,8) has the minimum sums of all *Arc Distance Metric* that has been performed. Hence, the algorithm picks this (1,2,3,7,8) as the winner for T_b, as noted by $Choice(T_b)$ on the diagram.

5.1.4 The Outline of Execution

To see the overall picture of the algorithm, it is important to note that the map-matching algorithm is executed per update, and has some steps which are illustrated by the flow chart figure 5.4. The first step is to get estimated location update from the navigation system. It will then check whether the hypotheses set are empty or not. Empty hypotheses set indicates either the system does the first update at its startup or that the system failed to match the reported location to any road segment for some times. The latter is because the algorithm prunes all "dead" (i.e. too improbable) hypotheses and will eventually left the system with no hypotheses at all (if there are no matches at all for some time).

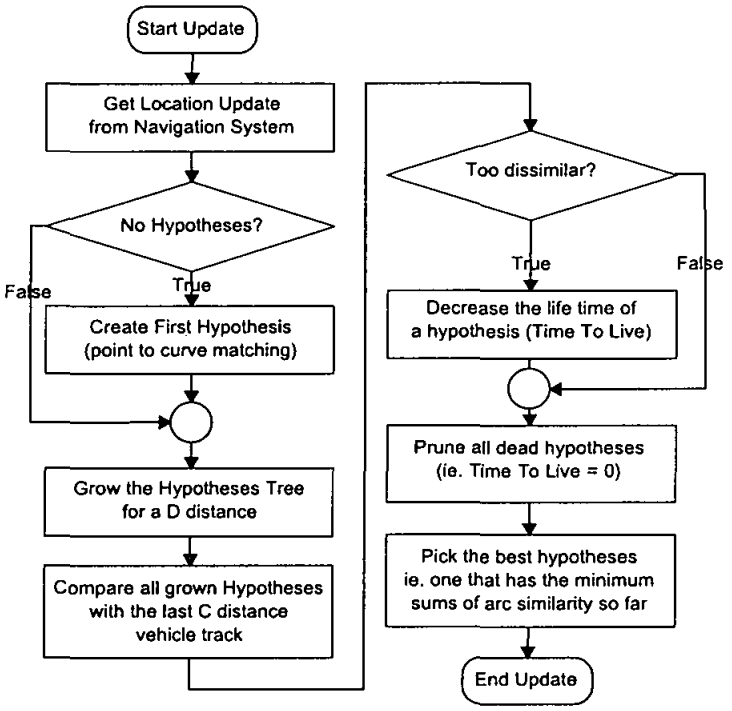


Figure 5.4: Flowchart of the algorithm

An empty hypotheses set caused by two conditions above will need an initial hypotheses generation process (First Guess or FG), which will be using another map-matching methods. The current configuration uses point-to-curve algorithm. It will list all road segments within a circular perimeter around the estimate vehicle location and find the nearest spot within that road segment. Of all these possible starting points, the system picked the nearest point to become the tentative best result.

If the hypotheses set $\mathcal{H}$ is not empty, the system will just grow it for approximately D distance. It does the growing by further tracing the adjacent road network for the approximate length of D, and to incorporate "good" candidates into the hypotheses set. After the growing, the system will compare all hypotheses with the last vehicle track for some distance C. The comparison is done using *Arc Distance Metric*. The details of this growing process is already discussed previously, leaving only the hypotheses reduction to be explained in the next section.

5.1.5 Detailed Data Structure and the Notation

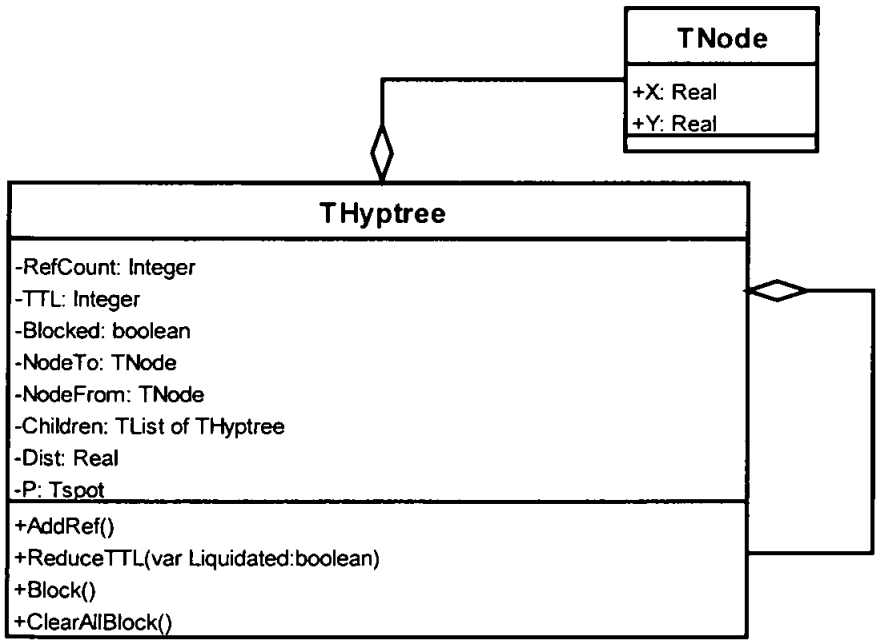


Figure 5.5: class diagram for hypothesis node

The hypotheses tree in the earlier sections, probably at the simplest level will be realized as an object of the class in Figure 5.2b. The figure represent a class (in the OOP terminology) whose instance would be able to contain a number of itself, which could be used to model our hypotheses tree. The root of that tree is already present on the tree structure itself, thus the term hypothesis node and hypothesis tree are actually refer to the same thing.

For further conceptual discussions we will use a simpler notation (compared to the UML class above) to represent the hypotheses node, by capturing the most important fields within the hypothesis node structure. Note that this notation will replace the

notation on Figure 5.2b for further discussions.

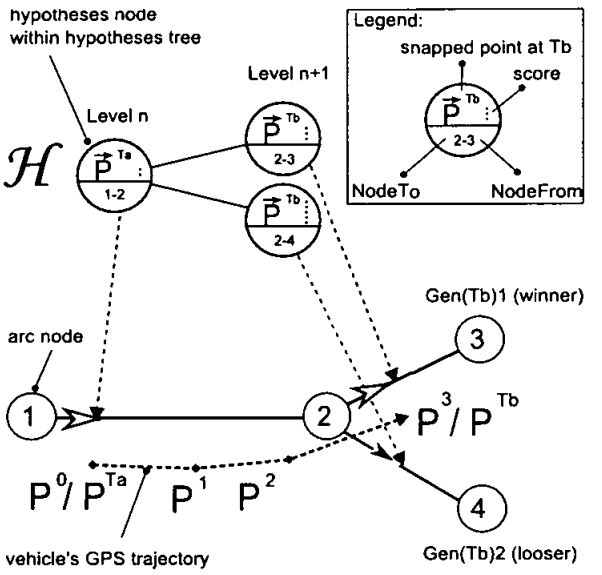


Figure 5.6: simpler notation for a hypothesis node

The Figure 5.1.5 shall explain this notation. The most important fields of a hypothesis node is the P, NodeTo, and NodeFrom field. P is the snapped point within an arc segment on the network representation. This arc segment is denoted by the combination of NodeTo and NodeFrom. Hypothesis H, for example, has $H = \{P = (100, 100), NodeTo = 2, NodeFrom = 1\}$. It means that, according to H, the vehicle travels from Node 1 to Node 2, and currently is located in (100,100), whatever the coordinate means. For the implemented simulator, the coordinate is merely pixels. This coordinate is, of course, already transformed into a pixel based representation from the latitude/longitude pair.

The next most important field is Score, which is the *Arc Distance Metric* score performed for that hypotheses. The calculation of *Arc Distance Metric* performed by evaluating the *full path* of that hypotheses node. By full path we mean that all of its ancestor are traced, thus giving us the route hypotheses which the last portion of it is to be compared with the last portion of vehicle trajectory. Figure 5.1.5 should illustrate these concepts clearly. The score in that figure is only represented as dots. The higher the dots or the score means the more *Arc Distance Metric* the node has, and it will be less likely selected as the winner. As stated in section 5.1.3, a winner is selected from the sum of score from the root to the leaf node.

The point P must lie within the straight line between *NodeTo* and *NodeFrom*. This is assuming that we are dealing with planar maps. This assumption is safe for small scale maps such as city maps, where straight line nearly approximate the great circle distance

(the shortest distance between two points on the earth's surface). For a much larger map a straight line may not be (depending on the projection) representing this great circle distance.

The main hypotheses tree contain a special root which do not contain any positional information. This special root is created at the initial program execution and serves as a single ancestor for every other generated positional hypotheses afterwards.

5.1.6 Hypotheses Reduction

The reason behind the reduction is because the algorithm presents an ever-growing and often explosive hypotheses tree problem. Leaving the hypotheses nodes intact would burden the system's memory and also the processing unit. The reduction algorithms are responsible for omitting the unnecessary nodes in the hypotheses tree. There are three implemented reduction algorithms that are implemented in this work as summarized in the following table:

Reduction Algorithm	Traits
Basic Pruning	Eliminates unlikely hypotheses before it is even attached. The most natural filtering and even necessary for the accuracy of the map-matching algorithm.
Compaction	Eliminates the redundant hypotheses within an arc. It gives maximum two hypotheses within an arc. Very useful for low velocity setting such as in slow traffic.
Merging	Merges very similar hypotheses branch into one physical branch, but still maintain the logical existence of them. Very useful for city setting which contains many similar roads (such as road pattern in blocks in a very ordered city)

Table 5.1: Reduction algorithms and their traits

5.1.7 Basic Pruning Technique

On this technique, the unnecessary nodes would be a set of unlikely hypotheses, i.e. the new hypotheses that is fail to score the necessary similarity, and also a set of infertile nodes, i.e. the hypotheses that is not growing for some time.

Every growth of hypotheses needs to be below some threshold of *Arc Distance Metric*, which mean that if the generated hypothesis is not good enough (badly dissimilar route),

it will not be attached to a growing point in the hypothesis tree. If certain hypothesis node failed to grow for some time, it can be inferred that such hypothesis node is not fertile, i.e. it then becomes an unnecessary node.

It is possible then to assign Time-To-Live (TTL) counter to each node of the hypothesis tree. The TTL counter is reduced for all "infertile", non-growing hypotheses. On the other hand, the TTL counter remains intact for a "fertile", growing hypotheses branch. When a particular branch has zero TTL, it will be pruned up to non-dead nodes. The experiment showed that this pruning algorithm successfully maintains the prospective nodes and reducing the number of hypotheses if the TTL is initially set to 5.

5.1.8 Hypotheses Compaction Technique

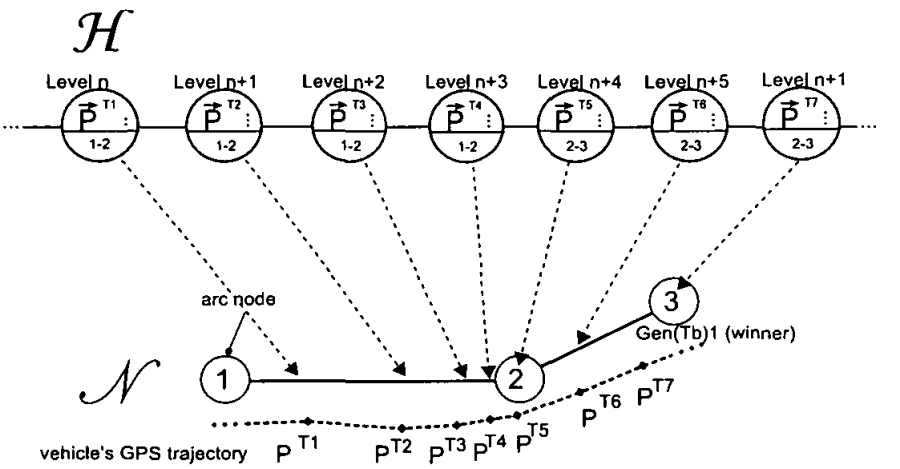
On this technique, the unnecessary nodes would be redundant hypotheses nodes within an arc. If the vehicle has low velocity then it is more likely that the grown hypotheses will still be on the same arc, only shifted or shortly spaced away. In this case not all nodes are necessary to represent the vehicle's journey.

The minimum number of nodes to represent a vehicle's journey is exactly one per road arc. Unfortunately the current algorithm architecture does not allow it to be realized in an easy way, that we can only cut the size only to two minimum hypotheses nodes per road arc. One of the node is always on the shape point or an arc node (arc node is not hypothesis node), and the other one is in-between the shape points. These two nodes are called the boundary nodes, and the nodes in-between these nodes are called internodes. The internodes are the removal target of the compaction algorithm.

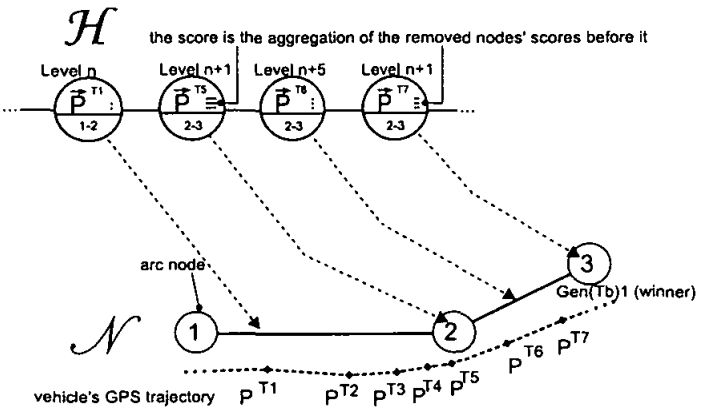
The removal of the internodes should not affect the scoring of the hypothesis, so there must be a kind of carry-forward (or backward) mechanism to bring the score to other unremoved hypothesis node.

It must be noted however, that the removal of hypotheses will inevitably affect (remove) the previous unnecessary snapping points, and thus preventing us to reconstruct the vehicle's time-to-time journey, unless if some vehicle track logging/recording schema (that are independent of the hypotheses tree) are employed.

The figure 5.7 shows the situation before and after compaction. Before compaction there are nine hypotheses nodes. As illustrated in 5.7a, the node of level n+1,n+2,and n+3 of this path are considered unnecessary. Their score is represented as the number of bars on the right side of the hypotheses nodes' circular symbol.Their *Arc Distance Metric* score is then carried forward to the node at level n+4. Hence their scores are summed (3x3+3=12 bars) and became the score at the node level n+4.



(a) Before compaction



(b) After compaction

Figure 5.7: Hypotheses compaction technique

5.1.9 Hypotheses Merging Technique

The motivation of this merging is based on the redundant partial hypotheses that must be maintained especially when there are route split and route join. Consider the road situation in Figure 5.8. Suppose the vehicle is traveling the route 1-2-3-5-6. Alternatively there is road route 1-2-4-5-6. The GPS record shows that both road route are indeed possible, so the algorithm would maintain both route in its hypotheses tree. Eventually these two route alternatives would meet in arc segment 5-6. Unfortunately the proposed algorithm knows nothing about this and still maintains the alternatives in two differing branches. Of course any new hypotheses nodes grown on one branch are also grown on the other branch, effectively drains the system resource.

This merging technique tries to reduce the physical node by eliminating many similar, joining hypotheses nodes and redirect all references to it into one selected physical node. All of the nodes are still logically preserved (with some small error as a result of small location difference or small hypothesis score difference), only that the reference to that node is redirected to the physical node. By physical node, we mean a hypothesis node that is allocated as an object occupying a chunk of memory, whereas logical node is only a reference (pointer) to other physical node.

The hypotheses nodes that are mergeable must satisfy these conditions:

1. The nodes are recently explored.

   A node is recently explored if it is attached to the hypothesis tree at the latest current update.

2. The nodes is considered similar. Two hypotheses nodes are similar if :

   a) the *Arc Distance Metric* score is less than some constant;

   b) The nodes are on the same arc and heading on the same direction;

   c) The euclidean distance between two hypotheses node are less than some constant;

If those conditions are met, the first found hypothesis node of the mergeable set will be retained and referenced, leaving the others to be freed. Any other references to the freed nodes will be redirected to the retained node. Thus the system may save a lot of memory and processing resources.

Managing such alias objects is quite error-prone, especially on objects creation or deletion (freeing). Reference Counting is a popular programming technique to help maintaining such sanity check [40] and thus also implemented here. It monitors how many objects are referencing another object using a variable stored in the refenced object.

In our implementation of reference counting, the system will be responsible for incrementing the counter variable of a hypothesis node (RefCount) whenever another reference

is pointing to that node. Later, upon the hypothesis node deletion by the function Safe-Free(), the system will first decrement the RefCount. The real deallocation happens only if the RefCount of the hypothesis node is zero after the decrement.

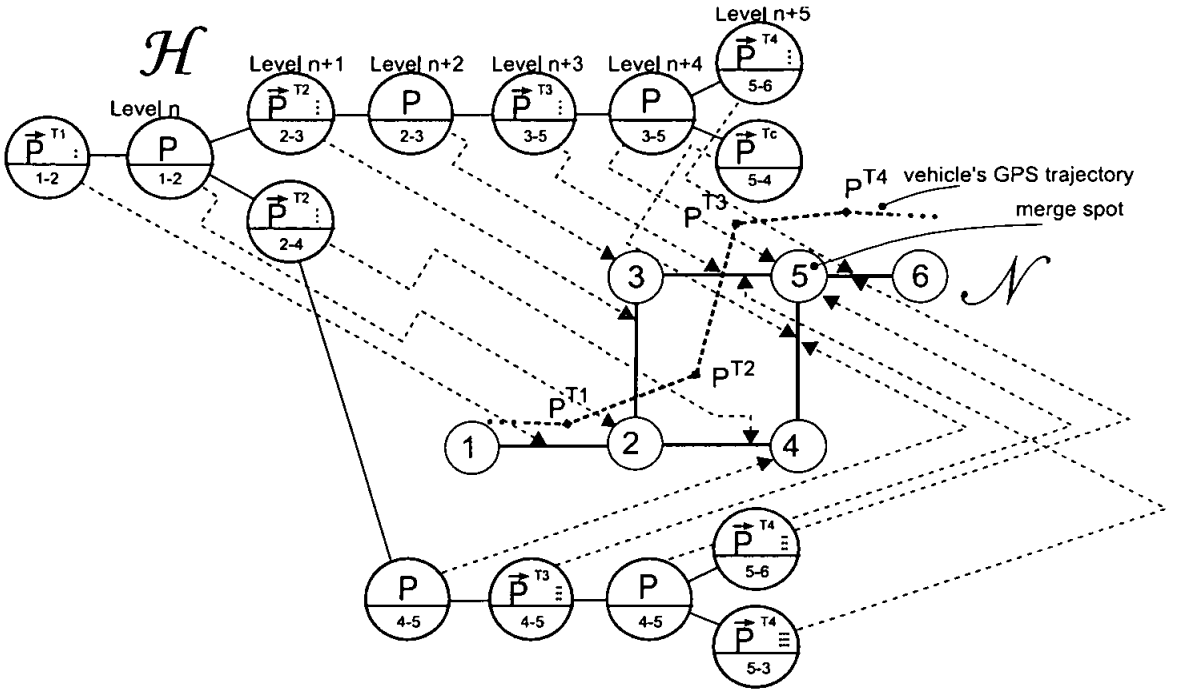


Figure 5.8: Regular hypotheses tree without merging

The figure 5.8 is also depicting the hypotheses tree before merging. The vehicle is assumed running from left to right through the reported vehicle's GPS trajectory. Thus it is quite probable that before the merging there are two main branches of hypotheses following the road structure. The branch splitted after T_1, giving two estimate snapping points at T_2, that are $\overrightarrow{P}^{T_2}$ that lies on arc 2-3, and the other one that lies on arc 2-4. Each of that branch are developing another hypotheses on its own, so that for $\overrightarrow{P}^{T_3}$ there are two hypotheses as well, either the one that lies on arc 3-5 and 4-5.

However the two branches eventually meets in the arc 5-6. Both branches are producing a similar hypotheses for the update at T_4. Both are producing $\overrightarrow{P}^{T_4}$ that lies on arc 5-6. The difference is that the upper branch that follows the route (1-2-3-5-6) also produce a hypothesis on arc 5-4 whereas the lower branch (1-2-4-5-6) produce a hypothesis on arc 5-3.

If the merging technique is used within the proposed algorithm, the hypotheses tree for such scenario will be like Figure 5.9. At the leaf, there is only one real hypotheses node created for $\overrightarrow{P}^{T_4}$ that lies on arc 5-6. This hypotheses is logically seen as two, for lower

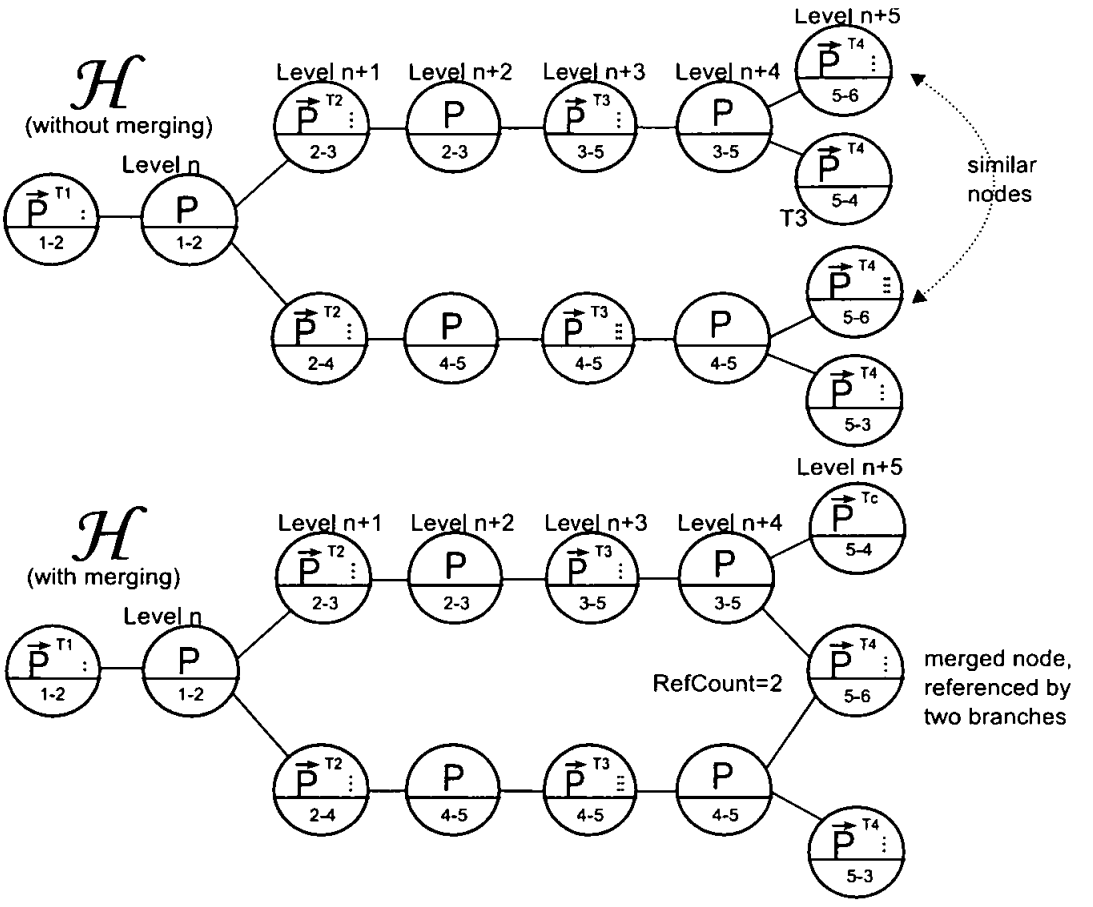


Figure 5.9: Hypotheses Tree with Merging Technique

branch and for upper branch. Further calculation of *Arc Distance Metric* and growing is not committed when one of the branch already done, since doing so will violate the model's integrity. Hence the memory is saved and the computing speed is also gained.

5.2 Implementation

This section is devoted to explain the implementation of our proposed map-matching algorithm. The implemented algorithms are presented in a pseudocode form that is derived from Object Pascal, but we use the procedural style pseudo-code to convey many primitives on the algorithm, even though the real implementation is using object-oriented paradigm. The reason is to add the readability of the pseudo-code introduced. We deviate from Object Pascal on the use of **for each** statement to replace the ordinary **for** statement. We feel that the former is clearer and much easier to understand.

Many other helper or very simple routines will not be explained further. These are called primitives and will be briefly explained right after they are introduced in the pseudo-code.

5.2.1 The Core Algorithms

The core algorithms includes the *Arc Distance Metric* metric calculation routines, a framework algorithm that manages the hypotheses in a tree data structure, and a merging hypotheses technique. The merging hypotheses technique is included because it is deeply intertwined with the core algorithm.

Update

The Update() procedure will be called after some specified intervals, e.g. 4 GPS epoch (four seconds). If it cannot find any hypotheses, then the first guessing process will take place. Rep is the vehicle's reported GPS location. It modifies the Guessed spot, by examining the hypotheses tree after growing it for LastDist distance. Rep and Guessed are of type TSpot, which is simply a structure that is consisted of X and Y location of vehicle within the map, after translated from the Latitude/Longitude position.

This procedure is considered high-level and needs many lower-level procedures / function to achieve its goal. The most important of all is TraverseTree(), which is responsible for traversing the whole hypotheses tree (Hyp) and then growing it for some distance.

The Update() procedure needs to initialize (clear) a stack of hypotheses tree when it is called. This stack is named Path and will be useful inside the TraverseTree() function to maintain the hypotheses nodes for every recursive traversal of the tree. This procedure is also responsible for incrementing the global variable Epoch, and works on global variable Hyp (the main hypotheses tree itself).

The RecentlyExploredNodes is a list containing the newly grown nodes of the current update. It is not updated directly on the Update() function, but will be updated within the Grow() function which will be called by TraverseTree() function.

CompactTree() is the procedure to do hypotheses compaction technique as explained in Section 5.1.8. The merging hypothesis technique is done inside the Grow(), which is called from TraverseTree() function. Both techniques use the blocking information of a hypothesis node. This blocking information is simply a boolean field which indicates whether a hypothesis tree is already accessed or not during the tree traversal within TraverseTree() and CompactTree(). The blocking information is particularly useful to avoid unnecessary (redundant) processing of multiple-referenced nodes introduced by the merging-technique.

Algorithm 3 Update() procedure to perform map-matching, called per several epochs.

```
procedure Update(Rep: TSpot; LastDist: Real; var Guessed: TSpot);
var
    Path: stack of hypothesis tree;
begin
    Epoch := Epoch + 1;
    if NumberOfNodes(Hyp) = 1 then FirstGuess(Rep,Hyp);
    Clear(Path);
    Clear(RecentlyExploredNodes);
    ClearAllBlocks(Hyp);
    BestResult := TraverseTree(Hyp,Path,LastDist);
    ClearAllBlocks(Hyp);
    CompactTree(Hyp,nil);
    Guessed := BestResult.Point;
end;
```

There are two primitives here that does not need to be explained in terms of pseudocode. These primitives are:

1. procedure Clear(L:List);

The pseudocode invoke this on Clear(RecentlyExploredNodes).

2. procedure Clear(P:Path);

The pseudocode invoke this on Clear(Path).

Both procedures will clear the respective container and thus no item is within the container after it executes. There are no object deletion on these procedures.

Initial Guessing Process (FirstGuess)

The FirstGuess() procedure is responsible for creating initial hypotheses from empty hypotheses set. This is done by enumerating all roads arc within the network (obtained from global variable Roads). For each arc segment it will find the nearest spot to the reported GPS position at that time. The calculation of the nearest segment and the nearest spot is done through the IsNearSegment() function.

The primitives for the pseudocode are:

1. function CreateNewTree(P:Point, FromNode:Node, ToNode:Node):THypTree

   Creating a new Tree and set the three most important member variable of it:

   The position P, FromNode and ToNode.

2. procedure AttachNewChild(Child:THypTree,Parent:THypTree);

Algorithm 4 First Guess Algorithm using Point To Curve Matching

```
procedure FirstGuess(Rep:TSpot; Tree: THypTree);
var
  I,J: Integer;
  R: TRoad;
  CDist: Real;
  Proj: TSpot;
  ChildTree: THypTree;
begin
  for each R in Roads do
  begin
    if VertexCount(R) = 2 then
    begin
      if IsNearSegment(R.FromNode,R.ToNode,Rep,CDist,Proj)
      then begin
        ChildTree := CreateNewTree(Proj, R.FromNode, R.ToNode);
        AttachNewChild(NewTree,Tree);
        ChildTree := CreateNewTree(Proj, R.ToNode, R.FromNode);
        AttachNewChild(NewTree,Tree);
      end;
    end else begin
      for J:=0 to VertexCount(R) - 2 do begin
      if IsNearSegment(GetVertexNode(J,R), GetVertexNode(J+1,R), Rep, CDist, Proj)
      then begin
        ChildTree := CreateNewTree(Proj, GetVertexNode(J,R), GetVertexNode(J+1,R));
        AttachNewChild(NewChild,Tree);
        ChildTree := CreateNewTree(Proj, GetVertexNode(J+1,R), GetVertexNode(J,R));
        AttachNewChild(ChildTree,Tree);
      end;
    end;
  end;
end;
```

Attaching the new children of Parent Tree by adding Child to the Children list of Parent Tree.

3. function VertexCount(Road):Integer;

Returning the number of vertices in this Road arc.

4. function GetVertexNode(I,Road):TNode

Returning the node that is the vertex at index I of the vertices list or this Road arc.

function IsNearSegment()

This helper function will return boolean value indicating whether the nearest point of a particular segment is considered near or far. A near segment means the nearest spot on the segment is lesser than some constant radius (CONST_FIRSTGUESS_RADIUS), and vice versa. To perform its job, the IsInside() function is needed.

Algorithm 5 Checking whether a segment has a quite near points

```
function IsNearSegment(N1,N2:TNode; F:TSpot; var D:Real; var PProj: TSpot): boolean;
var
    FromSpot, ToSpot: TSpot; begin
    FromSpot := NodeToSpot(N1);
    ToSpot := NodeToSpot(N2);
    IsInside(FromSpot,ToSpot, F, Dist, PProj);
    Result := (Dist <CONST_FIRSTGUESS_RADIUS);
end;
```

function IsInside(P1,P2,F:TSpot; var D:Real; var PProj: TSpot): boolean

This function checks whether the projection of F (pproj) lies inside the line segment constructed from P1 and P2. If it is, it will calculate the distance from F to that projection, Dist(PProj,F). Else, it will calculate the nearest of the two endpoints, i.e. the minimum of Dist(F,P1) and Dist(F,P2). The D variable will be set according to the distance calculated. The PProj variable will be set to the projected F within the line, if there is a projection.

The primitives called within this function are:

1. Vectorize(A,B:TSpot):TSpot

Will return a vector based on the difference between two points A and B.

Algorithm 6 Calculating the nearest point of a segment to another point outside

```
function IsInside(P1,P2,F:TSpot; var D:Real; var PProj: TSpot): boolean;
var
    MagnitudeASquared: Real;
    MagnitudeBSquared: Real;
    CosSquared: Real;
    A,B,C: TSpot;
    X,Y: Real;
    AdotB: Real;
begin
    MagnitudeASquared := Sqr(Dist(F,P1));
    MagnitudeBSquared := Sqr(Dist(P1,P2));
    A := Vectorize(P1,F);
    B := Vectorize(P1,P2);
    C := Vectorize(P2,F);
    AdotB := DotProduct(A,B);
    X := Abs(DotProduct(A,B));
    Y := Abs(DotProduct(C,B));
    Result := (X ≤ MagnitudeBSquared) and (Y ≤ MagnitudeBSquared));
    if Result then begin
        CosSquared := Sqr(AdotB)/(MagnitudeASquared * MagnitudeBSquared);
        Dist := Sqrt(Abs(MASquared*(1-CosSquared)));
        PProj := Addition(P1,CrossProduct( AdotB / MagnitudeBSquared, B ));
    end else begin
        Dist := Min(Dist(F,P1),Dist(F,P2));
        if Dist = Dist(F,P1) then PProj := P1 else PProj := P2;
    end;
end;
```

2. DotProduct(A,B:TSpot):Real
Will return the dot product between two vectors A and B.
3. CrossProduct(A,B:Tspot):TSpot
Will return the cross product between two vectors A and B.
4. Abs(X:Real):Real
Will return the positive values of X.
5. Sqr(X:Real):Real
This function returns the Square of X.
6. Addition(A,B):TSpot
This returns the addition of two vectors A and B.
7. Dist(A,B): Real
This function returns the euclidean distance between A and B.

The TraverseTree() function

This is a core function responsible for visiting every nodes within the hypotheses tree. After it has reached the leaves, it calls Grow() to explore and to grow the next nodes. The growth of hypotheses (how far the ahead tracing is performed) is limited by Dist variable. The path variable is a stack containing the hypotheses nodes (contains all previous snapping points) up to the current node (Tree variable).

The tree traversal is done using Depth First traversal. The basis of the recursion is when the leaf is reached (the number of children for the node is zero). This is the point where the Grow() is called. The recurrence happens when the node visited is not a leaf node. In the recurrence part, the Grow() will be called, and new hypotheses are added during this process. The Grow() will return the newly growned hypothesis and its score. Then the minimum score hypothesis (i.e the most similar to the partial vehicle track) is sought among all candidates. The candidates are the new hypotheses score, obtained from return values of the Grow() function for each of its children.

Below are the primitives used:

1. function CountChildren(Tree:THyptree):Integer;
   Returns the number of children of a hypothesis tree node.
   If the number of children is zero, it means that the node is a leaf.
2. function IsBlocked(Tree:THyptree):Boolean;
   Returns true if the node is blocked. Otherwise it is false.
3. procedure Block(Tree:THyptree);
   Will block a particular node.

4. procedure Push(Tree:THyptree; Stack:THypStack);
   This procedure will push Tree into Stack.
5. function Pop(Stack:THypStack):THyptree;
   This function returns the item that has been popped from Stack.
6. procedure SetLargeValue(var Score);
   An initializer for the score, will set into the largest possible value.
7. function IsGoodResult(Result):boolean;
   Returns true if the result's Score is lesser than a constant. Else it will return false.
8. procedure AddItemToList(Item,List);
   This procedure will add Item to the list as the last item.
9. procedure RechargeTimeToLive(Tree:THyptree);
   This procedure will set the (Time-to-Live) TTL field of a particular hypothesis tree to a constant.
10. procedure ReduceTimeToLive(Tree:THyptree);
    This procedure will decrement the TTL of Tree.
11. function IsZeroTTL(Tree:THyptree):boolean;
    This function returns true if the TTL is zero for a Tree.

The Grow() function

This function is responsible for growing the hypotheses tree according to the possible route ahead and the similarity of the route grown to the last portion of the reported vehicle trajectory. The function grows the hypotheses for the length of Dist. Therefore the recursion stop if the Dist is less than a (small) constant indicating it has grown enough length. This Dist variable is obtained from the estimation of the last traveling distance of the vehicle.

After it has stop growing for a branch, the calculation of *Arc Distance Metric* is performed. This is done by Compare() function. The results of this function is the score of the metric. If the score is lesser than CONST_MAX_HYPOTHESES_THRESHOLD then it is considered as a good hypothesis.

If the Dist is still large and the tree is not blocked, the hypotheses branch still needs to be grown. The ProduceChildren() is a function which produce a set of route ahead. For example, if there are road junctions ahead, then the produced set of route would be the node at the intersection. It will decrement Dist up to the length required to the intersection. On the next ProduceChildren() call (by the next recursive execution of Grow()), the generated set of route would be a set of possible routes alternatives at the

Algorithm 7 Hypotheses tree traversal

```
function TraverseTree(Tree: THyptree; Path:THypStack; Dist:Real): TSimMeasure;
var I: Integer;
  MinResult: TSimMeasure;
  Liquidated: Boolean;
  BadHypothesis: THyptree;
  BadList: list of THyptree;
begin
  if CountChildren(Tree)=0 then begin
    if IsBlocked(Tree) then
      Result := Grow(Tree, 0, Path, 0)
    else
      Result := Grow(Tree,Dist,Path,0);
  end else begin
    Block(Tree);
    Push(Tree,Path);
    SetLargeValue(MinResult.Score);
    for each Child in Tree.Children
      Result := TraverseTree(Child, Path, Dist);
      if (Result.Score ≤ MinResult.Score) or (I=0) then begin
        MinResult := Result;
      end;
      if IsGoodResult(Result) then
        RechargeTimeToLive(Child);
      else
        AddItemToList(Child,BadList);
      end;

      for each BadHypothesis in BadList
        ReduceTimeToLive(BadHypothesis);
        if IsZeroTTL(BadHypothesis) then begin
          RemoveChildOfTree(BadHypothesis,Tree);
          Free(BadHypothesis);
        end;
      end;
      MinResult.Score := MinResult.Score + Tree.Score;
      Result := MinResult;
      Pop(Path);
    end;
  end;
```

junction, for the modified Dist length.

Note that if the tree is already blocked, it means that that particular node is referenced by more than one branch and is already visited during the previous traversal. Thus, there is no need to calculate, expand or grow it further. Instead the function shall return with the existing Children.

The Compare() function

For the discussion of this function please refer also to Algorithm 9. This function prepares the calculation of *Arc Distance Metric* by trimming the possible route hypothesis and the trajectory altogether, so that both will have the same length. Next, it will invoke the *Arc Distance Metric* calculation function (CalcADM) with those trimmed line. This is the normal scenario, except on the initial guessing process, whereby FirstGuess() produce the initial hypotheses where the constructed routes only has single point and does not even form a line (hence we put a restriction that the minimum nodes count is two). In case where the minimum nodes less than two, it is not possible to invoke *Arc Distance Metric* calculation function and the function simply returns 0 (zero) for indicating perfect similarity.

LastDistance() is a simple utility to trim and convert a list (or stack) of hypotheses tree into a list of points. The resulting list of points are guaranteed to have the sum of distance (or total length) of CONST_BACKCOMPARE_DIST. CountNodes() is a function which count the nodes or element of a list or stack. *Traj* is a special list which contain the current vehicle trajectory reported from the GPS in form of hypotheses nodes.

The CalcADM() function

This function is for calculating the *Arc Distance Metric* that is already discussed in Chapter 4. It is important to note that a curve is modeled into a series of points. Within this function there are no more reference to a series of hypotheses nodes, but there are only series of points (taken from a list of hypotheses nodes by the LastDistance() function) represented by TPolyline type.

The first part of this function is to initialize some variables used. Sum is the variable which hold the sum of distance so far. Traced1 and Traced2 indicates whether the first curve and second curve has been traced, respectively. MaxDist is the maximum distance between two curves. IDX1 and IDX2 holds the active point within P1 and P2.

Next, P2 is translated so that the initial point coincide P1. The P3 is exactly P2 before translation. Then three cursors are set up, each positioned at the first point of the curves. Within the loop, the P1 and P2 are both being traced for DL at each iteration.

junction, for the modified Dist length.

Note that if the tree is already blocked, it means that that particular node is referenced by more than one branch and is already visited during the previous traversal. Thus, there is no need to calculate, expand or grow it further. Instead the function shall return with the existing Children.

The Compare() function

For the discussion of this function please refer also to Algorithm 9. This function prepares the calculation of *Arc Distance Metric* by trimming the possible route hypothesis and the trajectory altogether, so that both will have the same length. Next, it will invoke the *Arc Distance Metric* calculation function (CalcADM) with those trimmed line. This is the normal scenario, except on the initial guessing process, whereby FirstGuess() produce the initial hypotheses where the constructed routes only has single point and does not even form a line (hence we put a restriction that the minimum nodes count is two). In case where the minimum nodes less than two, it is not possible to invoke *Arc Distance Metric* calculation function and the function simply returns 0 (zero) for indicating perfect similarity.

LastDistance() is a simple utility to trim and convert a list (or stack) of hypotheses tree into a list of points. The resulting list of points are guaranteed to have the sum of distance (or total length) of CONST_BACKCOMPARE_DIST. CountNodes() is a function which count the nodes or element of a list or stack. *Traj* is a special list which contain the current vehicle trajectory reported from the GPS in form of hypotheses nodes.

The CalcADM() function

This function is for calculating the *Arc Distance Metric* that is already discussed in Chapter 4. It is important to note that a curve is modeled into a series of points. Within this function there are no more reference to a series of hypotheses nodes, but there are only series of points (taken from a list of hypotheses nodes by the LastDistance() function) represented by TPolyline type.

The first part of this function is to initialize some variables used. Sum is the variable which hold the sum of distance so far. Traced1 and Traced2 indicates whether the first curve and second curve has been traced, respectively. MaxDist is the maximum distance between two curves. IDX1 and IDX2 holds the active point within P1 and P2.

Next, P2 is translated so that the initial point coincide P1. The P3 is exactly P2 before translation. Then three cursors are set up, each positioned at the first point of the curves. Within the loop, the P1 and P2 are both being traced for DL at each iteration.

Algorithm 8 Hypotheses growing

```
function Grow(Tree: THyptree; Dist:Real; Path:THypStack; ExpansionLevels:
  Integer): TSimMeasure;
var
  I: Integer;
  Child,Similar: THypTree;
  Children: TList;
  R: Real;
begin
  Push(Tree,Path);
  if Dist ≤ CONST_NODE_PROXIMITY then begin
    R := Compare (Path);
    Tree.Score := R;
    Result.Score := R;
    Result.Good := R ≤ CONST_MAX_HYPOTHESES_THRESHOLD;
    Block(Tree);
  end else begin
    if not IsBlocked(Tree) then begin
      Children := ProduceChildren(Tree, Dist);
    end else begin
      Children := Tree.Children;
    end;
    SetLargeValue(MinResult);
    MinResult.Good := False;
    for each Child in Children do begin
      Result := Grow(Child, Child.Dist, Path, ExpansionLevels + 1);
      if Result.Score <MinResult.Score or (I=0) then
        MinResult := Result;
      if not IsBlocked(Tree) then begin
        if IsGood(Result) then
          if IsExistsSimilarNode(Child, SimilarChild) then begin
          AttachNewChild(SimilarChild,Tree);
        end else begin
          AttachNewChild(Child,Tree);
          AddItemToList(Child,RecentlyExploredNodes);
          AddRef(Child);
        end;
        end else begin
          SafeFree(Child);
        end;
      end;
    end;
  Result := MinResult;
  if not IsBlocked(Tree) then begin
    Free(Children);
    Block(Tree);
  end;
end;
Pop(Path); end;
```

Algorithm 9 Preparing two curves before calling *Arc Distance Metric* function

```
function Compare(Path:THypStack): Real;
var P1,P2: TPolyline;
begin
  if CountNodes(Path) ≥ 2 and CountNodes(Traj) ≥ 2 then begin
    P1 := LastDistance(Path,CONST_BACKCOMPARE_DIST);
    P2 := LastDistance(Traj,CONST_BACKCOMPARE_DIST);
    Result := CalcADM(p1,p2, CONST_DL); end else begin
      Result := 0;
    end;
  end;
```

It means that the cursors are advanced forward for a length DL, according to the shape of each curves. P3 is also has another cursor, and it will be used to obtain the maximum distance between P1 and P2 before translation. The loop continues until both P1 and P2 are fully traced. A curve is fully traced if the cursor already reached the last point of the curve, or the point index (IDX1 and IDX2) equals to the maximum point index within the curve. The Sum is then multiplied by the maximum distance found to find the result of *Arc Distance Metric*.

These primitives are used within this function:

1. function GetFirstPoint(P:TPolyline):TSpot;
   This function is to get the first point (zero index) within a list of points.
2. function Copy(P:TPolyline):TPolyline;
   This function returns a copy of a polyline.
3. function TranslatePoints(P:TPolyline; DX,DY: Real);TPolyline;
   This function returns a new polyline like P, only that it is translated according to DX and DY delta.
4. function Dist(P1,P2:TSpot):Real;
   This function returns the euclidean distance between P1 and P2.
5. function Max(A,B:Real):Real;
   This function returns the maximum of two real numbers.
6. function Length(P:TPolyline):Integer;
   This function returns the number of points comprising P.

Algorithm 10 Arc distance metric implementation

```
function CalcADM(P1,P2:TPolyline; DL: Real): Real;
var Pos1,Pos2,Pos3: TSpot;
  IDX1,IDX2: Integer;
begin
  Sum := 0;
  Traced1 := False;
  Traced2 := False;
  MaxDist := 0;
  DX := GetFirstPoint(P2).X-GetFirstPoint(P1).X;
  DY := GetFirstPoint(P2).Y-GetFirstPoint(P1).Y;
  P3 := Copy(P2);
  P2 := TranslatePoints(P2,DX,DY);
  Pos1 := GetFirstPoint(P1);
  Pos2 := GetFirstPoint(P2);
  IDX1 := 1;
  IDX2 := 1;
  repeat
    Trace(DL, P1, Pos1, IDX1);
    Traced1 := (IDX1 = Length(P1));
    Trace(DL, P2, Pos2, IDX2);
    Traced2 := (IDX2 = Length(P2));
    Trace(DL, P3, Pos3, IDX3);
    Sum := Sum + Dist(Pos1,Pos2) * DL;
    MaxDist := Max(MaxDist, Dist(Pos1,Pos3);
  until Traced1 and Traced2;
  Result := Sum * MaxDist;
end;
```

The Trace() function

This function advances the cursor Pos of a Curve for a length of DL. IDX is the index of the target vertex that is aimed by the cursor. Pos and IDX will be modified by this function as necessary.

Algorithm 11 Trace() function implementation

```
procedure Trace(DL:Real; Curve:TPolyline; var Pos:TSpot; var IDX:Integer);
var TV: TSpot;
  DTV, DX, DY, DELTAX, DELTAY: Real;
begin
  if (abs(DL) ≥ CONST_EPSILON) and (IDX ≠ Length(Curve)) then
    TV := Curve[IDX];
    DTV := Dist(TV, Pos);
    if DTV <DL then begin
      Pos := Curve[IDX];
      IDX := IDX + 1;
      Trace(DL-DTV,Curve,Pos,IDX);
    end else begin
      if abs(DTV) ≤ CONST_EPSILON) then begin
      DX := 0;
      DY := 0;
      end else begin
        DELTAX := TV.X - Pos.X;
        DELTAY := TV.Y - Pos.Y;
        DX := DL * DELTAX / DTV;
        DY := DL * DELTAY / DTV;
      end;
      Pos.X := Pos.X + DX;
      Pos.Y := Pos.Y + DY;
    end;
  end;
end;
```

5.2.2 The Hypotheses Compaction Algorithm

This algorithm is to implement the hypotheses compaction technique presented in 5.1.8. Specifically it will traverse the hypotheses tree and tries to eliminate all nodes which have all of these properties:

1. It has at least a parent.
2. It has exactly one child.
3. Either its parent or its child is in the same arc segment of the node.

The idea is that if a node has a same segment with the parent, or if that a node has a same arc segment with the child, then this node must lies in between a series of hypotheses nodes within the same arc segment. Thus this kind of node is not uniquely determine the route of the vehicle, and therefore may be eliminated with one note: the *Arc Distance Metric* score of this node must be carried to other node. We choose to carry it forward (has higher tree level), so that the eliminated node's Score field will be added to the child of it. The next thing to do is to make the parent node directly refer to the child of the eliminated node (GrandChild), which is similar to the technique used in the deletion of a linked list data structure.

5.3 Summary

This chapter discussed the development and design of our proposed framework map-matching algorithm. It further realized the concepts into a more concrete pseudocode, which is hopefully clarified the explanation. The next chapter (Chapter 6) will discuss the simulation software that is based on this algorithm. Figure 5.3 should summarize the system discussed in Chapter 4 and Chapter 5.

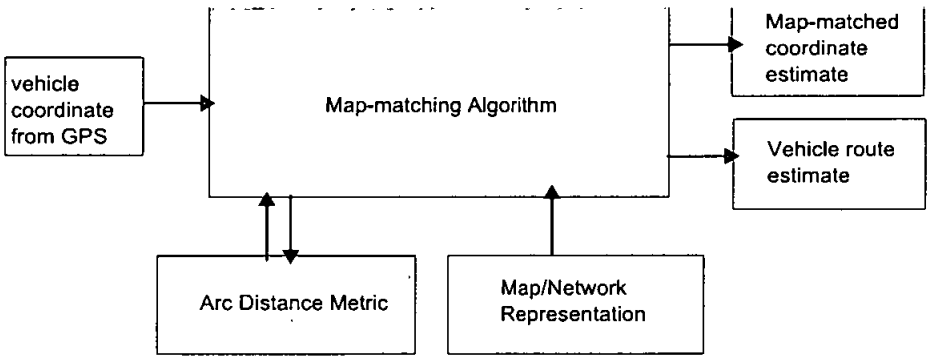


Figure 5.10: Map-matching algorithm and *Arc Distance Metric*

Algorithm 12 Hypotheses Compaction

```
procedure Compact(Tree, Parent:THypTree);
var GrandChild: THypTree;
  begin
    if CountChildren(Tree) >0 then begin
      if Parent = nil then begin
      for each Child in Tree.Children do begin
        Compact(Child, Tree);
      end;
    end else if IsSpecialRoot(Tree) or
    (CountRouteBranch(Tree) >1) or
    IsMerged(Tree) or
    ((not IsSameSegment(Tree,Parent)) and (not IsAllChildSameSegment(Tree)))
    then begin
      for each Child in Tree.Children do begin
        Compact(Child, Tree);
      end;
    end else begin
      RemoveChildFromParent(Tree,Parent);
      GrandChild := GetFirstChildren(Tree);
      GrandChild.Score := GrandChild.Score + Tree.Score;
      AttachNewChild(GrandChild,Parent);
      SafeFree(Tree);
      Compact(GrandChild,Parent);
    end;
  end;
end;
```

Chapter 6

The ViTracker Simulator

This chapter explains the design and implementation of a test bed environment for simulating map-matching process. The test bed proves to be very helpful on the development of the new map-matching algorithm, and deserves an explanation on this chapter. It is hoped that the developed environment will be usable for further research on the field.

6.1 Simulator Design

The simulator should resemble a GIS application, with the design requirements as follows:
1. The simulator must be able to playback the vehicle tracking data obtained from the field experiment with GPS receiver.
2. The simulator displays the vehicle, the road map, and the map-matching process during the playback. (Presentation aspects)
3. Other than playback capability, the simulator must also able to simulate the vehicle journey through some nodes, including its simulated, estimated GPS points. (GPS Simulation capability)
4. The simulator must implement *Arc Distance Metric* the framework algorithm for *Arc Distance Metric* and also basic map-matching algorithm using simple point-to-curve technique. (Test bed for Map-matching algorithms)
5. The simulator allows user to digitize road network from existing raster / bitmap background. (Map editing capability)

6.2 Implementation

The simulator is a Win32 application developed in Borland Delphi 7, and is able to run smoothly under 128MB Pentium III processors. The simulator is named ViTracker, to capture the Vehicle Tracking chore problem that it handles. For the following explanation, the reader should refer to the Figure 6.1 for the number references.

6.2.1 Playback of the recorded trajectory

A global time variable (6) is incremented periodically, according to the current simulation speed. Every simulator's second the vehicle's data is updated either with the recorded data (in the case of playback) or the simulated GPS point (in the case of GPS data simulation).

User may modify the speed of the simulation by dragging the speed control scrollbar (4). User could also pause the simulation, or do a step-by-step simulation as he/she wishes (5). The vehicle could also be centered automatically on the application window to avoid missing sight from the screen by ticking the checkbox (7). Lastly, the scale of the map is also adjustable by the zoom scrollbar (3). The global time is displayed in the time label (6). At the end of simulation, user might want to reset the overall simulation by pressing reset button (5).

6.2.2 Implementing the simulation capability

If the GPS data is simulated, the points are obtained from the vehicle's next waypoint, the current position of the vehicle, and some designed error factor. The vehicle goes from one waypoint to the next, until the end waypoint.

Some algorithms are used to reflect the typical case in the real scenario. For example, when the vehicle turns, it might be described from "hard turn" to "soft turn". The harder the road turn, the vehicle needs to slow down more. On the other side, the softer the road turn, the lesser the necessity to slow down. The algorithm measure the "hardness" of the turn ahead and calculates the speed reduction accordingly.

Another important feature is that of event architecture. An event indicates some change in the vehicle environment which affects the behavior (speed, acceleration, etc) of the vehicle. An example of it is traffic light event, where the vehicle will decelerate and stop for a number of seconds specified. Any event could be programmed to reflect the real scenario, and still easily integrated with the simulator.

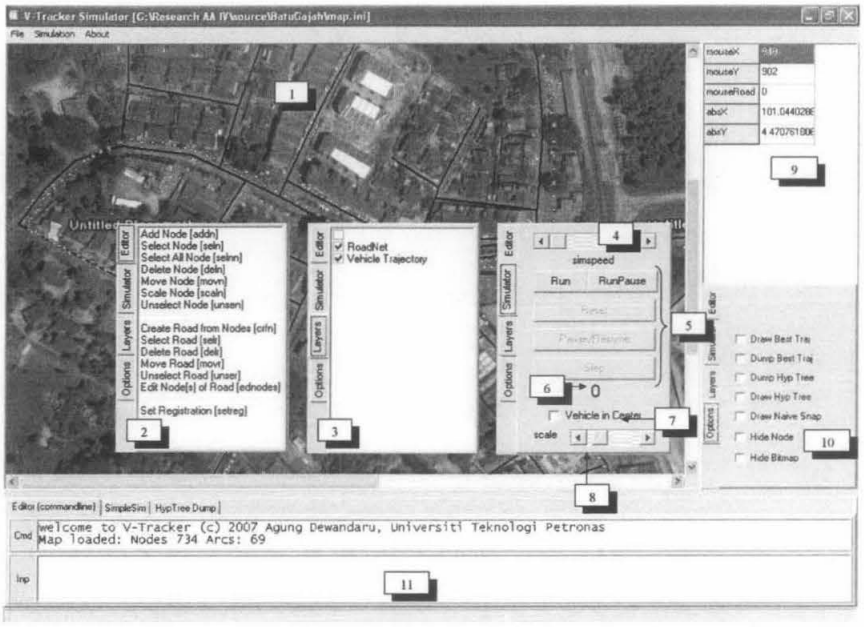


Figure 6.1: ViTracker snapshot and interface layout

6.2.3 User interface and presentation

ViTracker projects the vehicle and road network, superimposed with the background bitmaps, enabling the user to interpret clearly the process of map-matching. It could display the overall best, most likely trajectory, and also the tree data structure visualization. Many other options are available from these set of checkboxes (10).

ViTracker also has a layered architecture to make the selected road network to be invisible or not, depending on the user's choice. Each layer is a road network, i.e. a set of arcs that is loaded from a configuration file (3).

Lastly, the simulator is also augmented with the information about the X and Y pixel coordinates and its mapping to the real world latitude and longitude (9). Before it can do the mapping the user must register two points within the map and associate those points with the latitude and longitude of the represented spot.

6.2.4 Editing Map in ViTracker

ViTracker implements the digitizing capability, which enables the user to create nodes and arcs upon any raster map. Among the current implemented procedure are adding

nodes, selection of nodes, moving, and deletion. Also, user could create road arcs from selected nodes, delete, move, and edit the nodes that belong to a road arcs (2).

The editing UI resembles the AutoCAD style, blending the command line keyboard input with mouse interaction (11). This interaction style worked well to digitize the map which is used in the experiment (see Chapter 7).

6.2.5 Test Bed for map-matching Algorithms

During the execution of the simulation, ViTracker will do map-matching using the supported algorithm. ViTracker currently supports the basic point-to-curve map-matching and the proposed algorithm with the *Arc Distance Metric*. However, it has the infrastructure to support any other map-matching algorithms given the availability of the source code. The extension is currently at the source-code level, even though a dynamic-linked library seems to offer more security in the original program's code. Future development could include the binary level extension (DLL) and the implementation of the discrete Fréchet distance as the curve-to-curve distance metric.

6.2.6 Data File Specification

ViTracker works on three types of data file: the map file, a couple of layer files, and a simulation file. The map file would consists of a sequence of layer files that it uses.

Map File

The main purpose of a map file is to contain a list of arc nodes and a list of roads arcs. Every nodes has a pixel based coordinate, which is already transformed from the Latitude/Longitude pairs. Every road arc consists of a sequence of several nodes ID. Note that we relaxed the definition of arc as specified in Chapter 2.4. The difference is only on the connectivity of arcs. The relaxed definition allows the shape points to contain connection to other arc(s) whereas the stricter definition only allows the interconnection to happen only at the end-nodes. The effect of relaxing this is not significant in the result of accuracy because the hypotheses are stored in the form of arc segments.

Layer File

The layer file also contain a registration information for georeferencing purpose. It is fixing a logical node to a particular latitude/longitude coordinate based on two reference points. The program will calculate the necessary scaling for horizontal and vertical coordinate so

that both reference points has zero error. The effect on this registration is scoped on the layer level. It means that another layer would have another registration information.

Simulation File

Currently there are two modes of simulation: (1) GPS-based and (2) simulated-GPS. The GPS based simulation is the simulation mode that use real gps trajectory of the vehicle. It is stored as a special layer and thus has the same structure as the layer file that we have discussed. The recorded GPS points become the nodes of the layer and there is only one road arc which connects all of these nodes in a proper order.

The emulated GPS mode would need waypoints information. A virtual vehicle is then moved from a waypoint to the next waypoint, by adding some slow moving noise to mimic the real GPS signal. Additionally, there is an information about the events that might happened during the simulated course, for example traffic light or a traffic jam. This special event would modify the vehicle's speed attribute at a particular time.

Chapter 7

The Experiment and Results

An experiment had been conducted as a proof of concept of the proposed algorithm. The main objective of the experiment is to obtain vehicle trajectory in a pre-designated route in the form of Latitude/Longitude pairs. Next, it had to record when (the time) the vehicle enter an arc segment within the route. This will be the true route upon which the accuracy of the algorithm is measured.

7.1 Setup

7.1.1 The Vehicle Route

The area where the experiment should be done must be able to give a quite typical scenario for map-matching. We chose the area of Batu Gajah in Perak, Malaysia, situated around 4° 28' 6.39" N, 101° 2' 33.68" E. It is a small town of without high-risers but has considerable GPS signal obstacle along the route such as fly-over, and many two-three stories building siding the roads.

The road network was digitized using our simulator (ViTracker) from the combination of four image snapshots acquired from Google Earth. The digitizing process resulted 173 nodes, and 66 road arcs. We only use basic georeferencing technique without further calculated stretching. As a result, the digitized map is estimated to have worst accuracy of 29.6 m. This is a rather large value, but such inaccuracy is tolerable to test the performance of the algorithm, so any further calibration is not committed.

The two reference points that we used for georeferencing were estimated from the Google Earth coordinate on node 1 and node 49 seen on Figure 7.1. The Node 1 is 4°28' 35.380000" Latitude and 101°2' 28.900000" Longitude. This would be converted into a decimal notation of 4.47650°Lat 101.04136° Long. Similarly for node 49 it gives 4.47272°Lat 101.03783° Long.

Using this constructed map, the map accuracy is estimated around 29.6 meters. To find this, we got the estimated GPS benchmark A0987 location of: 4.4688945° Lat 101.0429425° Long. Then we got the digitized map, that benchmark location is: 4.4691399° Lat and 101.04284 ° Long. Then we use the distance calculator found in *http://jan.ucc.nau.edu/cvm-cgi-bin/latlongdist.pl* or *http://jan.ucc.nau.edu/ cvm/latlongdist.html* to find the distance between two points. It is 0.0296 km or 29.6 meters.

The planned route was picked from the digitized map. According to the plan, the car would be driven through that predetermined route. By visually comparing the digitized map and the GPS track and the time record, it would be known about what route taken by the vehicle and when. This would serve as the set of true values upon which the map-matching result of the algorithm is tested.

The roads sequence which compose the route was picked to give variety of GPS road obstacle scenarios and to give the close proximity of roads scenarios within for the road network. For example, the way under the flyover is a good scenario for both cases. First, it has multipath error possibility arise from the blocking of the flyover. Second, the flyover itself provides a close proximity choice over the road under it. A basic algorithm which do not take road connectivity into account supposedly fail on this test.

7.1.2 The Device Setup

We used the GARMIN e-Trex handheld GPS receiver to collect all estimated vehicle points location within the the overall route. During the setup, the epoch was set to one second so that it would give the maximum sequence of points. The datum used was the default WGS-84. The device was then mounted on the dashboard of the sedan that we would use.

7.2 The Experiment

The sky was bright, the weather was fine, and the handheld GPS receiver that we used reported varying accuracy, with the mode of 7 meters. We drove through predetermined route for 563 seconds, with reported total length 3.51 km, and the average speed 22.43 kmh. We used MapSource to pull and convert the recorded data into text format and later on imported that data to ViTracker. The data contains the information about the time, the estimate position (latitude and longitude) and other details which are listed in Appendix A.

The recorded GPS points are saved into a new layer within ViTracker. The layer configuration for the experiment is as follows:

1. Base layer. contains the aerial bitmap of Batu Gajah.
2. Road Network. contains all the (digitized) nodes and arcs which represents the road network in Batu Gajah.
3. Vehicle Track. Contains the list of all recorded vehicle points from the first second to the last second. The time that is recorded helps to determine the vehicle true location (arc). This will be used as a benchmark upon which the accuracy is measured.

7.3 The Results

The simulator was fed with the data, coupled with the new algorithm and also the basic map-matching algorithm (point-to-curve map-matching) as the benchmark. Out of the 140 system update (each four GPS ticks/seconds), the basic map-matching incorrectly matched 19 times (which is 86.43% accuracy) while the new algorithm presents no error but with 10 corrections. On the near real-time gross error it can be said that the algorithm performs better (92.86%), and on the final result the new algorithm presents perfect (100% accuracy) for road inferences. While this result cannot be over generalized due to the limited experiment but it proved that the algorithm works on a (quite typical) case with a considerable improvement of accuracy. The performance summary is shown in Table 7.1. Note that the Naive algorithm as the benchmark is implemented using the point-to-curve map-matching process.

Category	Naive	Proposed Algorithm
Total Errors	19	10
Total Prediction	140	140
Gross Accuracy	86.43%	92.86%
Corrected Err.	n/a	10
Net Accuracy	86.43%	100%

Table 7.1: Performance of the Algorithm

The digitized road network and the recorded vehicle route (obtained from GPS) are displayed in dots of Figure 7.1. The route started from the area (1) and ended in area (8). The direction of the vehicle is hinted by dashed arrows near the road arcs. In the area (2) the vehicle is speeding under the flyover. The vehicle will travel upon the flyover later. Area (3) has a heavier traffic. It should be noted that the driving rule in Malaysia is that all vehicle drives in the left side of the road, so when the vehicle wanted to turn right it has to wait for the opposite traffic first. This could be seen from the densely spaced dots in the figure near area (3).

The algorithm is fed with the recorded route and it gave the map-matched outcome in connected dots that lies exactly in the road-center lines. It can be seen that there is varying estimated vehicle track error, which is gradually changing over time. This is a direct effect of GPS and map inaccuracies resulting from digitizing and other map error. For example, the vehicle track around area (3) is shifted (or translated) to the southeast direction as compared to the map. However, in the vehicle track near area (6) the trends changed to north direction for a while. Nevertheless the new algorithm could cope with such inaccuracies pretty well.

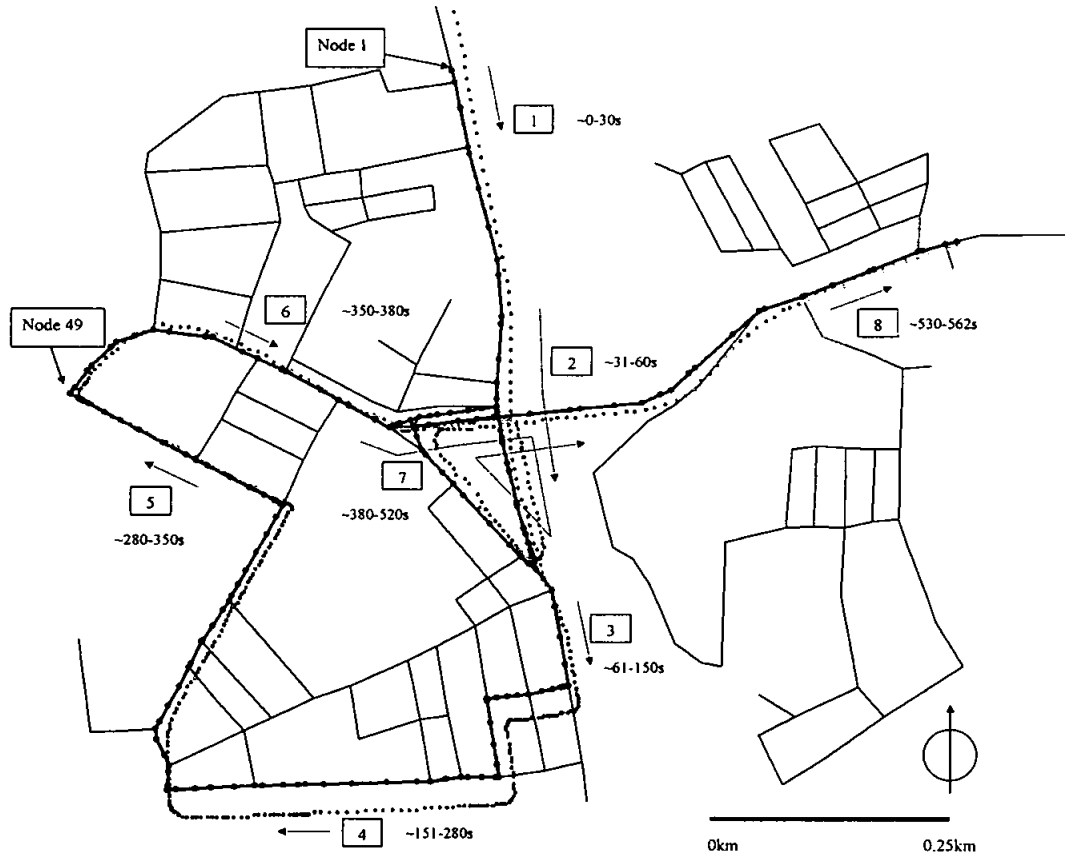


Figure 7.1: The Experiment Data and Map Matching Result

Chapter 8

Discussion

8.1 The *Arc Distance Metric* performance

The simulator implements the proposed map-matching algorithm, by using *Arc Distance Metric* as the primary means of curve-to-curve matching. This section discusses the *Arc Distance Metric* comparisons on a set of curves that was done during the simulation based on the experiment data. The following Figure 8.1 are a set of images reproduced from ViTracker which shows the curve similarity comparisons on $T = 105s$ (Fig. 8.1). They are grouped into seven clusters, each cluster comprised of three curves comparisons based from a road hypotheses generated from the arcs near the vehicle at that time.

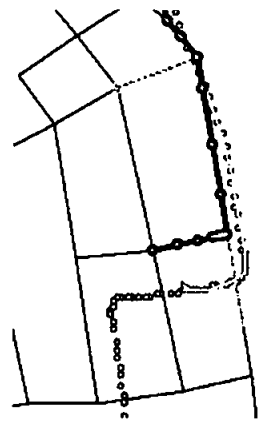


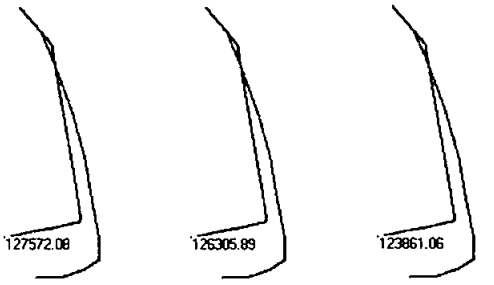
Figure 8.1: The road situation at $T = 105s$

The road curve hypotheses is indicated by a number near the end of it. The number shows the *Arc Distance Metric* score, which indicates the similarity between the road curve and the vehicle curve. This number is low when the road curve hypothesis is

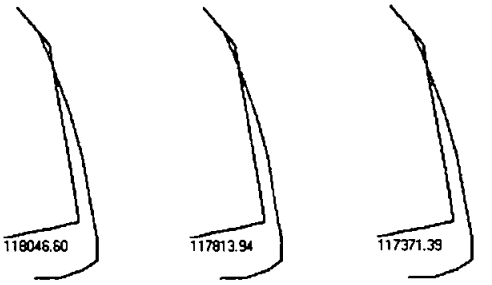
similar to the vehicle track. By visual observation it can be seen that this metric worked quite well for the problem, by giving lower score for dissimilar curve.

The design of *Arc Distance Metric* required that the metric consider both: the shape similarity of the curve and the lower Euclidean distance between them. These would make up the similarity of two curve i.e. equivalent curves (zero distance) are those who have equivalent shape and not spaced within any distance. But the design wanted to emphasize more on the shape rather than the distance, which is yet to be seen. Figure 8.2a display a rather dissimilar curve but very close proximity (i.e has low Euclidean distance). The *Arc Distance Metric* (δ_{adm}) for this pair reached more than 900,000 high. In contrast, the Figure 8.1e shows a rather similar pair of curves, but much more spaced away. Yet the *Arc Distance Metric* is around 500,000, which is still lower than the aforementioned figure. This shows a good compliance to the design criterion: dissimilar curves have more penalties than inter-curves Euclidean distance.

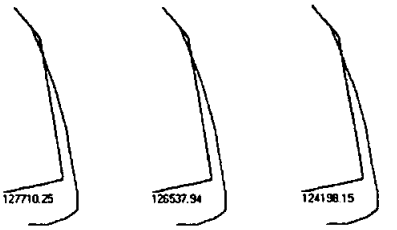
(a) 100%-90%-70% variants of road 1



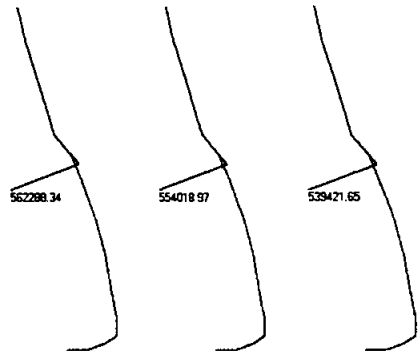
(b) 100%-90%-70% variants of road 2a



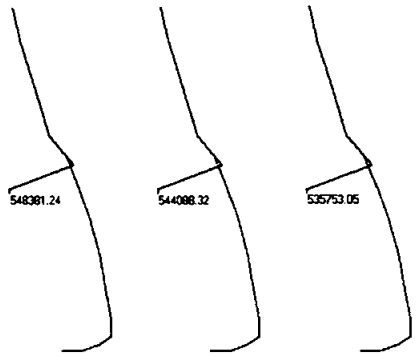
(c) 100%-90%-70% variants of road 2b



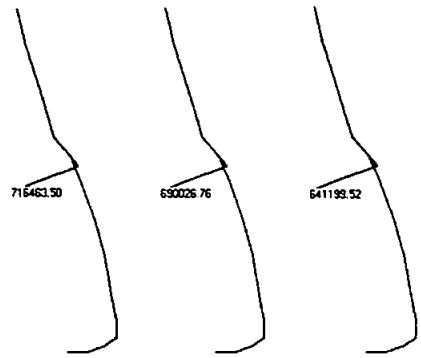
(d) 100%-90%-70% variants of road 2c



(e) 100%-90%-70% variants of road 3a



(f) 100%-90%-70% variants of road 3b



(g) 100%-90%-70% variants of road 3c

Figure 8.1: Curve similarity using *Arc Distance Metric*

Two of the three curves for each cluster seen are actually a slight modification from the original one. The modification lies on the distance between the last point and the point before that. Specifically, the last arc segment of the original road hypothesis (the original curve) is dilated into two variants: 90% length and 70% length. This will make three curves for each road hypothesis: two shorter curves and one normal curve (100% length). These available variants will be helpful to solve the *last distance estimation error*. This error will be explained in the next section.

8.2 Last Distance Estimation Error

At each update the algorithm grows the hypotheses by tracing the road arc connectivity ahead for some distance. The grown hypotheses are used to compare with the previous vehicle track. Here the problem is that the distance of the grow process is only an estimate, i.e. the sums of the point-to-point distances per GPS epoch.

The dependency on this single estimate variable would result either an estimate route which is too long, or an estimate route which is too short, since the GPS inaccuracies always produce a varying horizontal errors. Both of this has a negative implication on the accuracy, whereby the vehicle estimated route is too far ahead the vehicle, or lags behind the actual vehicle. If this situation get worse to some degree, the road arcs near the snapped vehicle point will not represent the actual road near the real vehicle, and would bring to a "no good match" condition, i.e. all the newly generated hypotheses are discarded. Hence, the hypotheses number will be decreasing since there is no nodes addition but there are a lot of nodes deletion(deleted by the pruning algorithm).

The implemented solution to this problem is to derive several variants, each having a modified distance. In the current implementation, it is done by multiplying 1.1 to the last distance obtained from the GPS and then applying a 100%-90%-70% last arc segment contraction. The multiplier caters for the lagged behind case (therefore it should be set to a number which will lengthen the last distance for some small percentage) while the contraction cares about too far ahead case. This technique will force the system to favor and pick the lowest *Arc Distance Metric* among those variants, therefore providing a chance of recovery for the position discrepancies based on the distance metric.

8.3 Refining the model

The working model of the algorithm uses only the locational information (latitude and longitude). This might be improved by adding the heading information (from the GPS

receiver) of the vehicle and use that information as a variable in measuring the similarity. The algorithm is quite general however, since it does not use many heuristics involving the movement of the vehicle. Hence, it might be a good candidate for solving other problem that presents multiple possibilities within a set of restricted constraints such as road network or grammar.

8.4 Backtracking and Correction Performance

The design of the proposed map-matching algorithm takes care about the possible backtracking, to cope with the incremental nature of the problem. According to the test result (see Figure 8.2), there are only eight corrections with the peak increment happens near simulation time 400 ($T = 400s$).

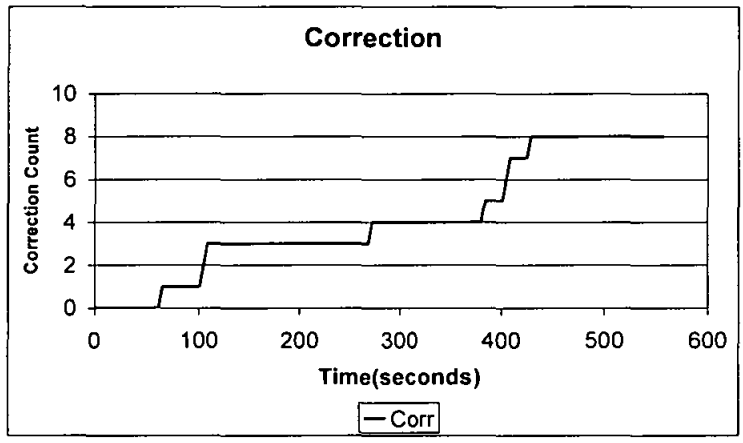


Figure 8.2: The Number of Correction vs Simulation Time

As seen on the Figure 8.3, the road situation near the vehicle at ($T = 400s$) is quite challenging for the map-matching . Figure 8.3a shows that the vehicle is facing many possible (three) roads ahead. The straight way (a) leads to the flyover; a slight to the left (b) is the road *beside* the flyover; while the right turn (c) is actually restricted by the traffic regulation. The road beside the flyover (b) will connect to the road that lies under the flyover, indicated by the vertical line.

The thick line with spaced dots indicates the predicted vehicle's track and the previous snapped points. There are two such lines on that figure: the horizontal on the left and

the vertical on the right. The vertical line is actually the previous track at simulation time between 31 and 60, and this is already the actual road taken by the vehicle (i.e. converged true hypothesis). Meanwhile the line on the left is relatively new ($T \geq 400$) hypothesis and therefore a bit volatile: it may change depending on the next information.

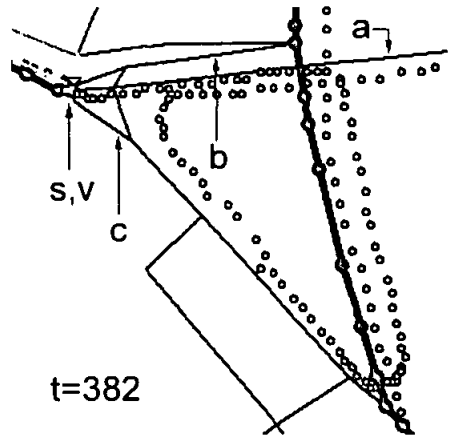
The hollow dots indicate the actual vehicle's track. The vehicle position itself at the time is indicated by a small triangle, and emphasized by an arrow with label "V". The snapped point on the road is indicated by the dots that always been constrained within a road arc, and emphasized by an arrow with label "S". If "V" is near "S" then it would be pointed by only one arrow with the label "S,V".

Referring to the Figure 8.3b, a manual inspection would suggest that later at that time($T = 417s$), the vehicle is most likely taking the flyover road arc. Hence, this is exactly the hypotheses inferred by the system, and the estimate vehicle line is updated to reflect this situation. However, this is not true. The actual road that was taken is the road beside the flyover (b), then the vehicle turn right and travel through the road *below* the flyover.

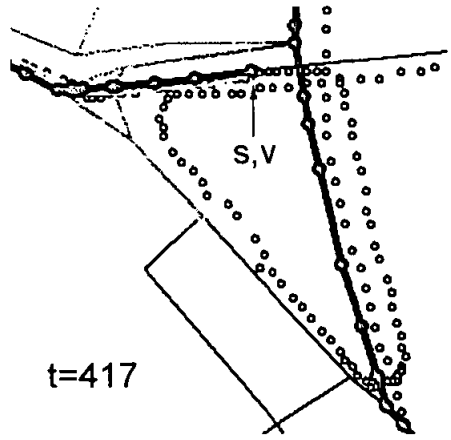
The lightly colored roads indicate that it is another probable hypothesis that is maintained by the system. The purpose is for the system to do backtracking, if it needs to do so. A road arc is considered as a hypothesis (lightly colored) if it is similar enough to the last vehicle track (i.e. below some *Arc Distance Metric* threshold). In this case, the road (b) and (c) is also lightly colored (note that we do not implement the road direction information yet, hence it is also consider road c as a valid hypothesis). This is because although road (b) and (c) are not the most likely hypothesis, they are not really bad on their *Arc Distance Metric* score (i.e. scoring a low enough distance as compared to the vehicle track), so it will be maintained as a probable hypotheses by the system.

At ($T = 429s$), a manual inspection would say that it is impossible for the vehicle to travel on the flyover, since there was a sharp right turn just now. This is also exactly what the system had suggested. The system performed an update (and grew the hypotheses as necessary) at that time and did evaluation on every possible hypotheses' score. It grew the hypothesis (a) further along the flyover; it also grew the hypothesis (b) into two branches: a sharp right turn and a sharp left turn. How far the system grows the hypotheses will be determined by how far the vehicle had moved from the last update.

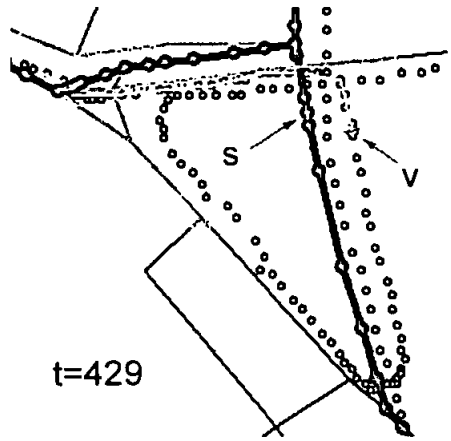
After it had evaluated the hypotheses set, it found that the previous best hypothesis (a) (the flyover) scored lower than the hypothesis (b). Hence the system backtracked and picked the road (b) followed by a sharp right turn.



(a) The vehicle before a complex junction



(b) Wrong inference



(c) Corrected inference

Figure 8.3: Example of backtracking in the experiment

8.5 Hypothesis Tree Reduction Techniques Results

The algorithm presents an ever-growing hypotheses tree problem, which will need to be pruned for its realistic applications. When the number of hypotheses gets too high, the system's resource could not process them all in a real-time fashion. It means that the simulator could only advance to the next second after it had spent more than one second processing. This condition called hypotheses explosion and it needs to be avoided. Thus it is very important issue to tackle for realistic applications of the algorithm.

There are four methods that are proposed in this thesis and all of them are already implemented (see Chapter 5.2). We will discuss the performance of basic pruning, hypotheses compaction and hypotheses merging techniques using the data set that is obtained from the experiment.

The first implemented and tested pruning algorithm is to assign Time-To-Live (TTL) counter to each node of the tree (basic pruning). The idea is basically that branches of hypotheses tree which are continuously growing (at each update) are considered to be decent hypotheses, while the static ones are bad. This basic pruning algorithm attached new hypotheses which are lower than a particular threshold and discarded the new hypotheses which are higher. Every time the tree grows, the TTL counter for all node within the tree is reduced, unless for a growing branch. When a particular branch has zero TTL, it will be pruned up to non-dead nodes.

As displayed in Figure 8.4, the basic pruning (BP) technique is able to keep the number of nodes manageable by the system. The number of nodes even decreasing in several occasions. This is clearly seen in simulation time 400-500. Without the pruning algorithm, the number of nodes to be managed is approximately twice (1345 nodes). With the basic pruning, it reduced down to 735 nodes.

The next implemented technique are the hypotheses compaction (C) technique. Using this technique many internodes hypotheses are successfully removed from the hypotheses tree without affecting the accuracy. The result presented in Figure 8.4 shows that this technique (BP+C) is able to further minimize the size of the tree (180 nodes).

The hypotheses merging (M) combined with other techniques (BP+C+M) could even minimize the real node counts. It works by the presence of similar, mergeable hypotheses. In the current experiment this technique combined with other technique only need 71 real nodes.

We used the combination of all these reduction technique for another simulated scenario. The simulated scenario is the condensed Batu Gajah map, which has much more nodes and roads (752 nodes, 93 Arcs). We used the recorded trajectory and expect a very heavy load because of the many similarities in the simulated network. Nevertheless

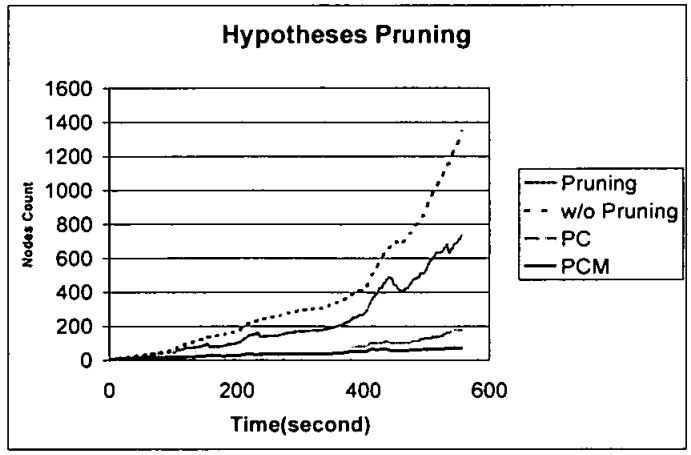


Figure 8.4: The Effect of Hypotheses Pruning

the merging technique proves to be very useful on this simulated scenario. It ended up with 16368 logical nodes, but with only 127 real nodes due to the merging.

8.5.1 Two Ways Internodes Hypotheses Generation Filter

The explosion of hypotheses easily happen if we allow the two-ways internodes hypotheses generation at each update. It is a good ability to handle assumption that the vehicle able to take a 180° at each update. Currently the algorithm is only allowed to branch the hypotheses only at the road junction, and the branching cannot go to the previous road. The solution to this is to implement a simple filter to detect the vehicle's reverse in the direction (e.g. 180° turn). It might be a fuzzy logic rule, neural networks, or even a simple algorithm based on vehicle's heading.

8.6 The Convergence of Hypotheses

Normally the algorithm will keep growing the hypotheses tree, unless there is huge discorrelation between the map and the vehicle track, whereby the hypotheses tree will be shrinking to a minimum size of zero hypotheses. This is the point where a new initial hypothesis is formed using point-to-curve algorithm.

In the normal state where the map and the vehicle track shows a good correlation, the map-matching take place as predicted, and the number of nodes on hypotheses tree will

be increasing. Normally the earlier hypotheses will be converging. In other words, there will be many nodes with only single child, representing the most probable hypotheses on the tree. Most of those nodes reside on the lower level of the tree (near the root). This is a situation where the hypotheses compaction technique is very useful.

8.7 Complexity analysis of the algorithm

The following discussion will make many references to Section 5.2.

We will take the analysis from the Update(). We will leave the FirstGuess() since it is performed in the initial part of execution only. The Update() contains two important functions (in terms of our developed algorithm) that is, TraverseTree() and CompactTree(). Other functions such as Clear() and ClearAllBlocks() are not hard and may have various implementation with various complexity, thus we will not discuss them.

8.7.1 Complexity of TraverseTree()

The CountChildren() could be implemented using variable lookup so it will not take more than O(1). Similarly, Block(), Push(), and SetLargeValue() are also quite simple and could be accomplished in O(1). The Grow() is recursive, and is using a tree, similar to a DFS search. We introduce the branching factor, that is, the average number of children of the hypotheses tree, represented as b. To simplify analysis, let us say that every tree expansion, every node will produce exactly b children, so the branching factor is b. Using that assumption, the number of nodes at depth n or less is $N = 1 + b + b^2 + \ldots + b^n$. This will reduce to $N = (b^{(n+1)} - 1)/(b - 1))$, which is O(b^n).

8.7.2 Complexity of Grow()

This function is an integral part of TraverseTree(), and is responsible for producing new children to be explored. This function is called only in the leaves of the hypotheses tree. There are b^n leaves to be grown, so if it is grown to add more m depth, then $N = b^n + b^{(n+1)} + \ldots + b^{(n+m)}$. Similar to the previous analysis, the complexity should be O($b^{(n+m)}$).

8.7.3 Complexity of CompactTree()

This algorithm performs compaction for every node in the hypotheses tree using recursive DFS technique. The number of nodes already described in the explanation above, and could also be reduced to O(b^n).

Chapter 9

Conclusions

9.1 Summary

The *arc similarity* distance and the new map-matching algorithm have been implemented. Both relies heavily on recursion, which make it somewhat limited by the stack and reduce its speed. The other consideration is about the explosion of hypotheses data, if the system allows two-ways internodes hypotheses generation at each update. It would be a good ability to handle assumption that the vehicle is able to take a 180° at each update. Currently the algorithm only allowed to branch the hypotheses tree only at the road junction, and the branches cannot refer to the previous road. The solution for this hypotheses explosion problem is to implement a simple detector of the vehicle's 180° turn, which will effectively limit the number of hypotheses. It might be a fuzzy logic rule, neural networks, or even a simple algorithm based on vehicle's heading. The hypotheses explosion problem can also be attacked using the developed hypotheses reduction algorithms. The hypotheses compaction and merging technique combined with the basic pruning is seen to work quite well in the data set.

In respect to the main objective, the simulation result showed a remarkable accuracy improvement due to the algorithm's ability to pick a better interpretation upon the vehicle track. On the second objective, it can be seen that the algorithm can infer the spot that the vehicle is located within the correct arc, albeit the several seconds latency given by the effect of the near real-time constraint. The algorithm is also able to form the meaningful travel route, as a consequence of the hypotheses generation that is always based on the real road network connectivity. This indicates the compliance to the third objective. Overall, while there are plenty of opportunities to improve the algorithm, it is observed that this model works and potentially capable to give a considerable improvement of accuracy.

9.2 Future Works

9.2.1 Model Improvement

While the current model is simple yet proven, a greater complexity is needed to enhance the ability for the algorithm to be adaptive. We recommend that the future model should add the road direction and road width as part of the network information. It should also consider the vehicle's direction to help improving the accuracy in a harder scenario.

The next improvement should be to enable the algorithm to detect reverse direction of the vehicle, and to incorporate the reverse road hypotheses (two ways internode hypotheses) if the vehicle's direction is reversing.

9.2.2 Implementation Improvement

The ViTracker could be made more useful if there is a binary extension mechanism, such as development of modular DLL to serve as "plug-ins". Furthermore, the addition of curve-to-curve matching technique such as Fréchet and Hausdorff would provide useful measure for a comparison or benchmark.

Bibliography

[1] WY Ochieng, M. Quddus, and RB Noland. MAP-MATCHING IN COMPLEX URBAN ROAD NETWORKS. *Brazilian Journal of Cartography (Revista Brasileira de Cartografia)*, 55(2):1–18, 2003.

[2] RR Joshi. Novel metrics for map-matching in in-vehicle navigation systems. *Intelligent Vehicle Symposium, 2002. IEEE*, 1, 2002.

[3] CE White, D. Bernstein, and AL Kornhauser. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C: Emerging Technologies*, 8(1):91–108, 2000.

[4] Jianyu (Jack) Zhou. A three-step general map matching method in the gis environment: Travel/transportation study perspective. In *UCGIS Summer Assembly 2005*, 2005.

[5] J.C. McCormac. *Surveying.* John Wiley and Sons, Inc., 2004.

[6] D. Michie, DJ Spiegelhalter, CC Taylor, and J. Campbell. *Machine learning, neural and statistical classification.* Ellis Horwood Upper Saddle River, NJ, USA, 1995.

[7] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification.* Wiley-Interscience, 2000.

[8] D. Bernstein and AL Kornhauser. An Introdution to Map Matching for Personal Navigation Assistants. *New Jersey TIDE Center (http://www.njtide.org/reports/mapmatchintro.pdf)*, 1996.

[9] M.S. Grewal, L.R. Weill, and A.P. Andrews. *Global Positioning Systems, Inertial Navigation, and Integration.* John Wiley and Sons, 2001.

[10] Ronald A. Brown. Instantaneous gps attitude determination. *Position Location and Navigation Symposium*, 1992.

[11] Guo-Shing Huang Jyh-Ching Huang. Development of gps based attitude determination algorithm. *IEEE transaction on Aerospace and Electronic System*, 1996.

[12] A. El-Rabbany. *Introduction to GPS: The Global Positioning System.* Artech House, 2002.

[13] G. Taylor. GIS and GPS integration and mobile handset positioning. *Web Information Systems Engineering (Workshops), 2002. Proceedings of the Third International Conference on*, pages 73–80, 2002.

[14] G. Taylor, G. Blewitt, D. Steup, S. Corbett, and A. Car. Road Reduction Filtering for GPS-GIS Navigation. *Transactions in GIS*, 5(3):193–207, 2001.

[15] JS Keates. *Understanding Maps.* Longman, 1996.

[16] John Campbell. *Map Use and Analysis.* McGraw Hill, 2001.

[17] Google Inc. last access 12-14-2007 6:36:16 PM. http://earth.google.com/userguide. 2007.

[18] BE Davis. *GIS: A Visual Approach.* Thomson Learning, 2001.

[19] J.S. Pyo, D.H. Shin, and T.K. Sung. Development of a map matching method using the multiple hypothesis technique. *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pages 23–27, 2001.

[20] M.A. Quddus, W.Y. Ochieng, L. Zhao, and R.B. Noland. A general map matching algorithm for transport telematics applications. *GPS Solutions*, 7(3):157–167, 2003.

[21] Tom M. Apostol. *Calculus vol 1, One-Variable Calculus, with an Introduction to Linear Algebra.* John Wiley and Sons, Inc, 1967.

[22] Sartaj Sahni Dinesh P. Mehta. *Handbook of Data Structures and Applications.* Chapman and Hall, CRC, 2005.

[23] D.E. Knuth. *The art of computer programming. Vol. 1: Fundamental algorithms.* Addison Wesley Longman, 1997.

[24] T.H. et. al Cormen. *Introduction to Algorithms.2nd Edition.* The MIT Press, 2001.

[25] D. Andersson and J. Fjällström. *Vehicle Positioning with Map Matching Using Integration of a Dead Reckoning System and GPS.* PhD thesis, Master Thesis, Linköping University, Sweden, 2004.

[26] M. Winter and G. Taylor. Modular Neural Network for Map-matched GPS Positioning. 2003.

[27] T. Kohonen. Statistical pattern recognition with neural networks: benchmarking studies. *IEEE International Conference on*, 1988.

[28] Salman Syed. *Development of Map-aided GPS algorithms for Vehicle Navigation in Urban Canyons.* PhD thesis, 2004.

[29] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On Map-matching Vehicle Tracking Data. *VLDB 31st Conference 2005. Proceedings.*, 2005.

[30] K. Ito. Encyclopedia of Mathematics, vol. 2, 1993.

[31] H. Alt, P. Brass, M. Godau, C. Knauer, and C. Wenk. Computing the Hausdorff distance of geometric patterns and shapes. *Discrete and Computational Geometry, Special Issue-The Goodman-Pollack-Festschrift*, 76, 2003.

[32] H. Alt, C. Knauer, and C. Wenk. Comparison of Distance Measures for Planar Curves. *Algorithmica*, 38(1):45–58, 2003.

[33] T. Eiter and H. Mannila. Computing discrete Fréchet distance. *Technische Universitat Wien Technical Report CD-TR*, 94:64, 1994.

[34] Helmut Alt and Michael Godau. Measuring the resemblance of polygonal curves. In *SCG '92: Proceedings of the eighth annual symposium on Computational geometry*, pages 102–109, New York, NY, USA, 1992. ACM.

[35] R. van Oostrum and R.C. Veltkamp. Parametric search made practical. *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 1–9, 2002.

[36] G. Taylor and G. Blewitt. Virtual differential GPS & road reduction filtering by map matching. *Proceedings of ION*, 99:1675–1684, 1999.

[37] G. Blewitt and G. Taylor. Mapping dilution of precision (mdop) and map matched gps. *International Journal of Geographical Information Science*, ISSN 1365-8816, 2002.

[38] D. Reid. An algorithm for tracking multiple targets. *Automatic Control, IEEE Transactions on*, 24(6):843–854, 1979.

[39] K. Ikeda F. Morisue. Evaluation of map-matching technique. *Vehicle Navigation and Information Systems Conference*, 1989.

[40] JJ Martin. Explicit Reference Counts. *Southeastcon '96*, 1996.

Appendices

Appendix A

The Vehicle trajectory derived from GPS

The following table are derived from Garmin GPS receiver at 7/9/2007 10:10:43 AM using WGS84 datum. The header "Time" is the simulation time. "Lat" and "Long" are latitudes and longitudes, respectively. "Alt" is for altitude.

"LL" is the leg length, "LTime" is the leg time, "LS" is the leg speed, "LCourse" is the heading of the vehicle. These four measures is for indicating the point-to-point line deltas (leg) in terms of its length, the time taken, and the course.

Time	Lat	Long	Time2	Alt(ft)	LL(ft)	LTime	LS(mph)	LCourse
1	N4.47656	E101.04152	10:32:55	92	40	00:00:01	27	169°
2	N4.47644	E101.04154	10:32:56	90	48	00:00:01	32	171°
3	N4.47633	E101.04158	10:32:57	90	42	00:00:01	29	158°
4	N4.47622	E101.04160	10:32:58	90	40	00:00:01	27	169°
5	N4.47611	E101.04162	10:32:59	90	40	00:00:01	27	169°
6	N4.47601	E101.04165	10:33:00	90	40	00:00:01	27	169°
7	N4.47590	E101.04165	10:33:01	90	39	00:00:01	27	180°
8	N4.47579	E101.04167	10:33:02	92	40	00:00:01	27	169°
9	N4.47568	E101.04169	10:33:03	92	40	00:00:01	27	169°
10	N4.47558	E101.04171	10:33:04	90	40	00:00:01	27	169°
11	N4.47547	E101.04173	10:33:05	90	40	00:00:01	27	169°
12	N4.47536	E101.04175	10:33:06	89	40	00:00:01	27	169°
13	N4.47526	E101.04177	10:33:07	89	40	00:00:01	27	169°
14	N4.47513	E101.04180	10:33:08	86	48	00:00:01	32	171°
15	N4.47502	E101.04182	10:33:09	84	40	00:00:01	27	169°
16	N4.47489	E101.04184	10:33:10	84	48	00:00:01	32	171°

17	N4.47476	E101.04188	10:33:11	82	50	00:00:01	34	162°
18	N4.47465	E101.04190	10:33:12	84	40	00:00:01	27	169°
19	N4.47453	E101.04192	10:33:13	84	48	00:00:01	32	171°
20	N4.47440	E101.04195	10:33:14	86	48	00:00:01	32	171°
21	N4.47429	E101.04197	10:33:15	86	40	00:00:01	27	169°
22	N4.47416	E101.04201	10:33:16	86	50	00:00:01	34	162°
23	N4.47405	E101.04203	10:33:17	86	40	00:00:01	27	169°
24	N4.47392	E101.04205	10:33:18	87	48	00:00:01	32	171°
25	N4.47382	E101.04208	10:33:19	87	40	00:00:01	27	169°
26	N4.47371	E101.04210	10:33:20	87	40	00:00:01	27	169°
27	N4.47360	E101.04210	10:33:21	87	39	00:00:01	27	180°
28	N4.47350	E101.04212	10:33:22	87	40	00:00:01	27	169°
29	N4.47337	E101.04212	10:33:23	87	47	00:00:01	32	180°
30	N4.47326	E101.04212	10:33:24	87	39	00:00:01	27	180°
31	N4.47315	E101.04212	10:33:25	87	39	00:00:01	27	180°
32	N4.47304	E101.04212	10:33:26	87	39	00:00:01	27	180°
33	N4.47294	E101.04212	10:33:27	87	39	00:00:01	27	180°
34	N4.47283	E101.04212	10:33:28	87	39	00:00:01	27	180°
35	N4.47274	E101.04212	10:33:29	87	31	00:00:01	21	180°
36	N4.47266	E101.04212	10:33:30	87	31	00:00:01	21	180°
37	N4.47255	E101.04212	10:33:31	89	39	00:00:01	27	180°
38	N4.47247	E101.04212	10:33:32	89	31	00:00:01	21	180°
39	N4.47236	E101.04212	10:33:33	89	39	00:00:01	27	180°
40	N4.47227	E101.04212	10:33:34	89	31	00:00:01	21	180°
41	N4.47219	E101.04212	10:33:35	90	31	00:00:01	21	180°
42	N4.47208	E101.04214	10:33:36	90	40	00:00:01	27	169°
43	N4.47199	E101.04216	10:33:37	90	32	00:00:01	22	166°
44	N4.47189	E101.04218	10:33:38	90	40	00:00:01	27	169°
45	N4.47180	E101.04220	10:33:39	92	32	00:00:01	22	166°
46	N4.47169	E101.04223	10:33:40	92	40	00:00:01	27	169°
47	N4.47161	E101.04227	10:33:41	93	35	00:00:01	24	154°
48	N4.47152	E101.04229	10:33:42	93	32	00:00:01	22	166°
49	N4.47144	E101.04231	10:33:43	93	32	00:00:01	22	166°
50	N4.47135	E101.04233	10:33:44	93	32	00:00:01	22	166°
51	N4.47126	E101.04235	10:33:45	95	32	00:00:01	22	166°
52	N4.47120	E101.04238	10:33:46	95	25	00:00:01	17	162°
53	N4.47111	E101.04240	10:33:47	95	32	00:00:01	22	166°

54	N4.47105	E101.04242	10:33:48	95	25	00:00:01	17	162°
55	N4.47098	E101.04244	10:33:49	95	25	00:00:01	17	162°
56	N4.47092	E101.04246	10:33:50	95	25	00:00:01	17	162°
57	N4.47086	E101.04248	10:33:51	95	25	00:00:01	17	162°
58	N4.47079	E101.04253	10:33:52	95	28	00:00:01	19	146°
59	N4.47073	E101.04255	10:33:53	95	25	00:00:01	17	162°
60	N4.47066	E101.04259	10:33:54	95	28	00:00:01	19	146°
61	N4.47060	E101.04261	10:33:55	95	25	00:00:01	17	162°
62	N4.47053	E101.04263	10:33:56	95	25	00:00:01	17	162°
63	N4.47047	E101.04265	10:33:57	97	25	00:00:01	17	162°
64	N4.47041	E101.04268	10:33:58	97	25	00:00:01	17	162°
65	N4.47034	E101.04270	10:33:59	98	25	00:00:01	17	162°
66	N4.47028	E101.04270	10:34:00	98	24	00:00:01	16	180°
67	N4.47021	E101.04272	10:34:01	98	25	00:00:01	17	162°
68	N4.47015	E101.04272	10:34:02	100	24	00:00:01	16	180°
69	N4.47008	E101.04274	10:34:03	100	25	00:00:01	17	162°
70	N4.47002	E101.04274	10:34:04	100	24	00:00:01	16	180°
71	N4.46998	E101.04276	10:34:05	100	18	00:00:01	12	154°
72	N4.46991	E101.04276	10:34:06	100	24	00:00:01	16	180°
73	N4.46985	E101.04278	10:34:07	100	25	00:00:01	17	162°
74	N4.46980	E101.04278	10:34:08	100	16	00:00:01	11	180°
75	N4.46978	E101.04280	10:34:09	100	11	00:00:01	7.5	135°
76	N4.46976	E101.04280	10:34:10	98	8	00:00:01	5.3	180°
77	N4.46974	E101.04280	10:34:11	100	8	00:00:01	5.3	180°
78	N4.46974	E101.04280	10:34:12	100	0	00:00:01	0	0°
79	N4.46974	E101.04280	10:34:13	98	0	00:00:01	0	0°
80	N4.46974	E101.04280	10:34:14	98	0	00:00:01	0	0°
81	N4.46974	E101.04280	10:34:15	98	0	00:00:01	0	0°
82	N4.46974	E101.04280	10:34:16	100	0	00:00:01	0	0°
83	N4.46974	E101.04280	10:34:17	100	0	00:00:01	0	0°
84	N4.46974	E101.04280	10:34:18	101	0	00:00:01	0	0°
85	N4.46974	E101.04280	10:34:19	101	0	00:00:01	0	0°
86	N4.46974	E101.04280	10:34:20	101	0	00:00:01	0	0°
87	N4.46974	E101.04280	10:34:21	101	0	00:00:01	0	0°
88	N4.46974	E101.04280	10:34:22	101	0	00:00:01	0	0°
89	N4.46974	E101.04280	10:34:23	101	0	00:00:01	0	0°
90	N4.46972	E101.04280	10:34:24	101	8	00:00:01	5.3	180°

91	N4.46972	E101.04280	10:34:25	101	0	00:00:01	0	0°
92	N4.46972	E101.04280	10:34:26	101	0	00:00:01	0	0°
93	N4.46970	E101.04280	10:34:27	101	8	00:00:01	5.3	180°
94	N4.46968	E101.04280	10:34:28	103	8	00:00:01	5.3	180°
95	N4.46968	E101.04280	10:34:29	103	0	00:00:01	0	0°
96	N4.46965	E101.04278	10:34:30	103	11	00:00:01	7.5	225°
97	N4.46965	E101.04278	10:34:31	103	0	00:00:01	0	0°
98	N4.46963	E101.04276	10:34:32	105	11	00:00:01	7.5	225°
99	N4.46963	E101.04272	10:34:33	103	16	00:00:01	11	270°
100	N4.46961	E101.04270	10:34:34	103	11	00:00:01	7.5	225°
101	N4.46961	E101.04265	10:34:35	103	16	00:00:01	11	270°
102	N4.46961	E101.04263	10:34:36	103	8	00:00:01	5.3	270°
103	N4.46959	E101.04259	10:34:37	103	17	00:00:01	12	243°
104	N4.46959	E101.04255	10:34:38	103	16	00:00:01	11	270°
105	N4.46959	E101.04253	10:34:39	103	8	00:00:01	5.3	270°
106	N4.46959	E101.04250	10:34:40	101	8	00:00:01	5.3	270°
107	N4.46959	E101.04246	10:34:41	101	16	00:00:01	11	270°
108	N4.46957	E101.04244	10:34:42	98	11	00:00:01	7.5	225°
109	N4.46957	E101.04240	10:34:43	97	16	00:00:01	11	270°
110	N4.46957	E101.04235	10:34:44	97	16	00:00:01	11	270°
111	N4.46957	E101.04233	10:34:45	97	8	00:00:01	5.3	270°
112	N4.46955	E101.04229	10:34:46	97	17	00:00:01	12	243°
113	N4.46955	E101.04227	10:34:47	97	8	00:00:01	5.3	270°
114	N4.46955	E101.04223	10:34:48	97	16	00:00:01	11	270°
115	N4.46955	E101.04220	10:34:49	97	8	00:00:01	5.3	270°
116	N4.46955	E101.04218	10:34:50	97	8	00:00:01	5.3	270°
117	N4.46955	E101.04214	10:34:51	97	16	00:00:01	11	270°
118	N4.46953	E101.04210	10:34:52	100	17	00:00:01	12	243°
119	N4.46955	E101.04212	10:34:53	98	11	00:00:01	7.5	45°
120	N4.46953	E101.04210	10:34:54	98	11	00:00:01	7.5	225°
121	N4.46950	E101.04210	10:34:55	98	8	00:00:01	5.3	180°
122	N4.46948	E101.04208	10:34:56	98	11	00:00:01	7.5	225°
123	N4.46946	E101.04208	10:34:57	98	8	00:00:01	5.3	180°
124	N4.46942	E101.04210	10:34:58	98	18	00:00:01	12	154°
125	N4.46938	E101.04210	10:34:59	98	16	00:00:01	11	180°
126	N4.46931	E101.04212	10:35:00	97	25	00:00:01	17	162°
127	N4.46927	E101.04212	10:35:01	97	16	00:00:01	11	180°

128	N4.46923	E101.04212	10:35:02	97	16	00:00:01	11	180°
129	N4.46918	E101.04214	10:35:03	97	18	00:00:01	12	154°
130	N4.46914	E101.04214	10:35:04	97	16	00:00:01	11	180°
131	N4.46908	E101.04214	10:35:05	97	24	00:00:01	16	180°
132	N4.46903	E101.04216	10:35:06	97	18	00:00:01	12	154°
133	N4.46899	E101.04216	10:35:07	97	16	00:00:01	11	180°
134	N4.46892	E101.04216	10:35:08	98	24	00:00:01	16	180°
135	N4.46888	E101.04216	10:35:09	98	16	00:00:01	11	180°
136	N4.46884	E101.04218	10:35:10	100	18	00:00:01	12	154°
137	N4.46880	E101.04216	10:35:11	100	18	00:00:01	12	206°
138	N4.46877	E101.04214	10:35:12	100	11	00:00:01	7.5	225°
139	N4.46875	E101.04212	10:35:13	101	11	00:00:01	7.5	225°
140	N4.46873	E101.04208	10:35:14	101	17	00:00:01	12	243°
141	N4.46873	E101.04205	10:35:15	101	8	00:00:01	5.3	270°
142	N4.46873	E101.04203	10:35:16	101	8	00:00:01	5.3	270°
143	N4.46873	E101.04199	10:35:17	101	16	00:00:01	11	270°
144	N4.46871	E101.04195	10:35:18	101	17	00:00:01	12	243°
145	N4.46871	E101.04190	10:35:19	101	16	00:00:01	11	270°
146	N4.46871	E101.04186	10:35:20	101	16	00:00:01	11	270°
147	N4.46871	E101.04182	10:35:21	101	16	00:00:01	11	270°
148	N4.46871	E101.04177	10:35:22	101	16	00:00:01	11	270°
149	N4.46871	E101.04173	10:35:23	101	16	00:00:01	11	270°
150	N4.46871	E101.04167	10:35:24	103	23	00:00:01	16	270°
151	N4.46871	E101.04162	10:35:25	103	16	00:00:01	11	270°
152	N4.46869	E101.04156	10:35:26	105	25	00:00:01	17	252°
153	N4.46869	E101.04150	10:35:27	105	23	00:00:01	16	270°
154	N4.46869	E101.04143	10:35:28	106	23	00:00:01	16	270°
155	N4.46869	E101.04135	10:35:29	108	31	00:00:01	21	270°
156	N4.46869	E101.04128	10:35:30	108	23	00:00:01	16	270°
157	N4.46867	E101.04120	10:35:31	108	32	00:00:01	22	256°
158	N4.46867	E101.04113	10:35:32	108	23	00:00:01	16	270°
159	N4.46867	E101.04105	10:35:33	109	31	00:00:01	21	270°
160	N4.46867	E101.04096	10:35:34	109	31	00:00:01	21	270°
161	N4.46867	E101.04087	10:35:35	109	31	00:00:01	21	270°
162	N4.46867	E101.04079	10:35:36	109	31	00:00:01	21	270°
163	N4.46867	E101.04070	10:35:37	111	31	00:00:01	21	270°
164	N4.46865	E101.04062	10:35:38	111	32	00:00:01	22	256°

165	N4.46865	E101.04051	10:35:39	111	39	00:00:01	27	270°
166	N4.46865	E101.04042	10:35:40	111	31	00:00:01	21	270°
167	N4.46865	E101.04034	10:35:41	111	31	00:00:01	21	270°
168	N4.46865	E101.04025	10:35:42	111	31	00:00:01	21	270°
169	N4.46865	E101.04019	10:35:43	111	23	00:00:01	16	270°
170	N4.46862	E101.04014	10:35:44	112	17	00:00:01	12	243°
171	N4.46862	E101.04012	10:35:45	112	8	00:00:01	5.3	270°
172	N4.46862	E101.04010	10:35:46	112	8	00:00:01	5.3	270°
173	N4.46862	E101.04008	10:35:47	112	8	00:00:01	5.3	270°
174	N4.46862	E101.04006	10:35:48	112	8	00:00:01	5.3	270°
175	N4.46862	E101.04002	10:35:49	112	16	00:00:01	11	270°
176	N4.46862	E101.03999	10:35:50	112	8	00:00:01	5.3	270°
177	N4.46862	E101.03995	10:35:51	112	16	00:00:01	11	270°
178	N4.46862	E101.03989	10:35:52	112	23	00:00:01	16	270°
179	N4.46862	E101.03984	10:35:53	112	16	00:00:01	11	270°
180	N4.46862	E101.03980	10:35:54	112	16	00:00:01	11	270°
181	N4.46862	E101.03976	10:35:55	112	16	00:00:01	11	270°
182	N4.46862	E101.03971	10:35:56	112	16	00:00:01	11	270°
183	N4.46862	E101.03965	10:35:57	112	23	00:00:01	16	270°
184	N4.46862	E101.03961	10:35:58	114	16	00:00:01	11	270°
185	N4.46860	E101.03954	10:35:59	114	25	00:00:01	17	252°
186	N4.46860	E101.03948	10:36:00	114	23	00:00:01	16	270°
187	N4.46860	E101.03941	10:36:01	114	23	00:00:01	16	270°
188	N4.46860	E101.03935	10:36:02	114	23	00:00:01	16	270°
189	N4.46860	E101.03931	10:36:03	114	16	00:00:01	11	270°
190	N4.46860	E101.03924	10:36:04	114	23	00:00:01	16	270°
191	N4.46860	E101.03920	10:36:05	114	16	00:00:01	11	270°
192	N4.46860	E101.03916	10:36:06	114	16	00:00:01	11	270°
193	N4.46860	E101.03909	10:36:07	114	23	00:00:01	16	270°
194	N4.46860	E101.03905	10:36:08	116	16	00:00:01	11	270°
195	N4.46860	E101.03901	10:36:09	116	16	00:00:01	11	270°
196	N4.46860	E101.03896	10:36:10	116	16	00:00:01	11	270°
197	N4.46860	E101.03892	10:36:11	116	16	00:00:01	11	270°
198	N4.46862	E101.03888	10:36:12	116	17	00:00:01	12	297°
199	N4.46865	E101.03884	10:36:13	116	17	00:00:01	12	297°
200	N4.46867	E101.03881	10:36:14	116	11	00:00:01	7.5	315°
201	N4.46869	E101.03881	10:36:15	116	8	00:00:01	5.3	0°

202	N4.46871	E101.03879	10:36:16	116	11	00:00:01	7.5	315°
203	N4.46873	E101.03879	10:36:17	116	8	00:00:01	5.3	0°
204	N4.46875	E101.03879	10:36:18	116	8	00:00:01	5.3	0°
205	N4.46880	E101.03879	10:36:19	116	16	00:00:01	11	0°
206	N4.46882	E101.03879	10:36:20	116	8	00:00:01	5.3	0°
207	N4.46886	E101.03879	10:36:21	116	16	00:00:01	11	0°
208	N4.46888	E101.03879	10:36:22	116	8	00:00:01	5.3	0°
209	N4.46892	E101.03879	10:36:23	116	16	00:00:01	11	0°
210	N4.46897	E101.03879	10:36:24	116	16	00:00:01	11	0°
211	N4.46901	E101.03877	10:36:25	117	18	00:00:01	12	334°
212	N4.46905	E101.03877	10:36:26	117	16	00:00:01	11	0°
213	N4.46910	E101.03877	10:36:27	117	16	00:00:01	11	0°
214	N4.46912	E101.03877	10:36:28	117	8	00:00:01	5.3	0°
215	N4.46916	E101.03877	10:36:29	116	16	00:00:01	11	0°
216	N4.46918	E101.03877	10:36:30	116	8	00:00:01	5.3	0°
217	N4.46923	E101.03877	10:36:31	117	16	00:00:01	11	0°
218	N4.46929	E101.03877	10:36:32	117	24	00:00:01	16	360°
219	N4.46933	E101.03877	10:36:33	117	16	00:00:01	11	360°
220	N4.46938	E101.03877	10:36:34	117	16	00:00:01	11	0°
221	N4.46942	E101.03877	10:36:35	117	16	00:00:01	11	360°
222	N4.46946	E101.03879	10:36:36	117	18	00:00:01	12	26°
223	N4.46950	E101.03879	10:36:37	119	16	00:00:01	11	0°
224	N4.46955	E101.03881	10:36:38	119	18	00:00:01	12	26°
225	N4.46961	E101.03884	10:36:39	119	25	00:00:01	17	18°
226	N4.46965	E101.03886	10:36:40	119	18	00:00:01	12	26°
227	N4.46970	E101.03888	10:36:41	117	18	00:00:01	12	26°
228	N4.46976	E101.03892	10:36:42	119	28	00:00:01	19	34°
229	N4.46980	E101.03894	10:36:43	117	18	00:00:01	12	26°
230	N4.46987	E101.03896	10:36:44	117	25	00:00:01	17	18°
231	N4.46991	E101.03899	10:36:45	117	18	00:00:01	12	26°
232	N4.46995	E101.03903	10:36:46	117	22	00:00:01	15	45°
233	N4.47000	E101.03905	10:36:47	117	18	00:00:01	12	26°
234	N4.47004	E101.03907	10:36:48	117	18	00:00:01	12	26°
235	N4.47011	E101.03911	10:36:49	117	28	00:00:01	19	34°
236	N4.47015	E101.03914	10:36:50	117	18	00:00:01	12	26°
237	N4.47019	E101.03916	10:36:51	117	18	00:00:01	12	26°
238	N4.47026	E101.03920	10:36:52	116	28	00:00:01	19	34°

239	N4.47030	E101.03922	10:36:53	117	18	00:00:01	12	26°
240	N4.47036	E101.03926	10:36:54	116	28	00:00:01	19	34°
241	N4.47041	E101.03929	10:36:55	116	18	00:00:01	12	26°
242	N4.47047	E101.03931	10:36:56	114	25	00:00:01	17	18°
243	N4.47051	E101.03933	10:36:57	114	18	00:00:01	12	26°
244	N4.47056	E101.03935	10:36:58	112	18	00:00:01	12	26°
245	N4.47060	E101.03937	10:36:59	112	18	00:00:01	12	26°
246	N4.47064	E101.03939	10:37:00	112	18	00:00:01	12	26°
247	N4.47066	E101.03941	10:37:01	112	11	00:00:01	7.5	45°
248	N4.47068	E101.03944	10:37:02	112	11	00:00:01	7.5	45°
249	N4.47073	E101.03946	10:37:03	112	18	00:00:01	12	26°
250	N4.47077	E101.03948	10:37:04	112	18	00:00:01	12	26°
251	N4.47079	E101.03950	10:37:05	112	11	00:00:01	7.5	45°
252	N4.47083	E101.03952	10:37:06	111	18	00:00:01	12	26°
253	N4.47088	E101.03954	10:37:07	111	18	00:00:01	12	26°
254	N4.47092	E101.03956	10:37:08	111	18	00:00:01	12	26°
255	N4.47096	E101.03959	10:37:09	111	18	00:00:01	12	26°
256	N4.47101	E101.03961	10:37:10	111	18	00:00:01	12	26°
257	N4.47105	E101.03965	10:37:11	111	22	00:00:01	15	45°
258	N4.47109	E101.03967	10:37:12	109	18	00:00:01	12	26°
259	N4.47114	E101.03969	10:37:13	109	18	00:00:01	12	26°
260	N4.47118	E101.03971	10:37:14	109	18	00:00:01	12	26°
261	N4.47122	E101.03974	10:37:15	109	18	00:00:01	12	26°
262	N4.47126	E101.03976	10:37:16	109	18	00:00:01	12	26°
263	N4.47131	E101.03980	10:37:17	111	22	00:00:01	15	45°
264	N4.47135	E101.03982	10:37:18	111	18	00:00:01	12	26°
265	N4.47139	E101.03984	10:37:19	109	18	00:00:01	12	26°
266	N4.47144	E101.03987	10:37:20	108	18	00:00:01	12	26°
267	N4.47148	E101.03989	10:37:21	109	18	00:00:01	12	26°
268	N4.47152	E101.03991	10:37:22	111	18	00:00:01	12	26°
269	N4.47156	E101.03993	10:37:23	111	18	00:00:01	12	26°
270	N4.47159	E101.03997	10:37:24	109	17	00:00:01	12	63°
271	N4.47161	E101.03997	10:37:25	111	8	00:00:01	5.3	0°
272	N4.47163	E101.03995	10:37:26	111	11	00:00:01	7.5	315°
273	N4.47163	E101.03995	10:37:27	111	0	00:00:01	0	0°
274	N4.47165	E101.03993	10:37:28	111	11	00:00:01	7.5	315°
275	N4.47165	E101.03991	10:37:29	112	8	00:00:01	5.3	270°

276	N4.47167	E101.03989	10:37:30	112	11	00:00:01	7.5	315°
277	N4.47167	E101.03984	10:37:31	112	16	00:00:01	11	270°
278	N4.47169	E101.03982	10:37:32	116	11	00:00:01	7.5	315°
279	N4.47171	E101.03978	10:37:33	116	17	00:00:01	12	297°
280	N4.47171	E101.03976	10:37:34	116	8	00:00:01	5.3	270°
281	N4.47174	E101.03971	10:37:35	117	17	00:00:01	12	297°
282	N4.47176	E101.03967	10:37:36	117	17	00:00:01	12	297°
283	N4.47178	E101.03963	10:37:37	117	17	00:00:01	12	297°
284	N4.47180	E101.03961	10:37:38	117	11	00:00:01	7.5	315°
285	N4.47182	E101.03954	10:37:39	119	25	00:00:01	17	288°
286	N4.47184	E101.03950	10:37:40	119	17	00:00:01	12	297°
287	N4.47186	E101.03946	10:37:41	119	17	00:00:01	12	297°
288	N4.47189	E101.03941	10:37:42	119	17	00:00:01	12	297°
289	N4.47193	E101.03935	10:37:43	120	28	00:00:01	19	304°
290	N4.47195	E101.03931	10:37:44	120	17	00:00:01	12	297°
291	N4.47197	E101.03924	10:37:45	122	25	00:00:01	17	288°
292	N4.47201	E101.03920	10:37:46	122	22	00:00:01	15	315°
293	N4.47204	E101.03914	10:37:47	122	25	00:00:01	17	288°
294	N4.47206	E101.03909	10:37:48	122	17	00:00:01	12	297°
295	N4.47210	E101.03903	10:37:49	122	28	00:00:01	19	304°
296	N4.47212	E101.03896	10:37:50	122	25	00:00:01	17	288°
297	N4.47214	E101.03892	10:37:51	120	17	00:00:01	12	297°
298	N4.47219	E101.03886	10:37:52	122	28	00:00:01	19	304°
299	N4.47221	E101.03881	10:37:53	122	17	00:00:01	12	297°
300	N4.47225	E101.03875	10:37:54	123	28	00:00:01	19	304°
301	N4.47227	E101.03868	10:37:55	123	25	00:00:01	17	288°
302	N4.47229	E101.03864	10:37:56	125	17	00:00:01	12	297°
303	N4.47234	E101.03858	10:37:57	127	28	00:00:01	19	304°
304	N4.47236	E101.03851	10:37:58	127	25	00:00:01	17	288°
305	N4.47238	E101.03845	10:37:59	128	25	00:00:01	17	288°
306	N4.47242	E101.03841	10:38:00	128	22	00:00:01	15	315°
307	N4.47244	E101.03834	10:38:01	130	25	00:00:01	17	288°
308	N4.47247	E101.03830	10:38:02	130	17	00:00:01	12	297°
309	N4.47249	E101.03823	10:38:03	130	25	00:00:01	17	288°
310	N4.47253	E101.03819	10:38:04	130	22	00:00:01	15	315°
311	N4.47255	E101.03815	10:38:05	131	17	00:00:01	12	297°
312	N4.47257	E101.03811	10:38:06	131	17	00:00:01	12	297°

313	N4.47259	E101.03808	10:38:07	131	11	00:00:01	7.5	315°
314	N4.47262	E101.03804	10:38:08	131	17	00:00:01	12	297°
315	N4.47264	E101.03800	10:38:09	131	17	00:00:01	12	297°
316	N4.47264	E101.03798	10:38:10	131	8	00:00:01	5.3	270°
317	N4.47264	E101.03798	10:38:11	131	0	00:00:01	0	0°
318	N4.47266	E101.03796	10:38:12	131	11	00:00:01	7.5	315°
319	N4.47266	E101.03796	10:38:13	131	0	00:00:01	0	0°
320	N4.47266	E101.03796	10:38:14	131	0	00:00:01	0	0°
321	N4.47266	E101.03793	10:38:15	133	8	00:00:01	5.3	270°
322	N4.47266	E101.03793	10:38:16	133	0	00:00:01	0	0°
323	N4.47266	E101.03793	10:38:17	133	0	00:00:01	0	0°
324	N4.47266	E101.03793	10:38:18	133	0	00:00:01	0	0°
325	N4.47266	E101.03793	10:38:19	133	0	00:00:01	0	0°
326	N4.47266	E101.03793	10:38:20	134	0	00:00:01	0	0°
327	N4.47266	E101.03793	10:38:21	134	0	00:00:01	0	0°
328	N4.47266	E101.03793	10:38:22	134	0	00:00:01	0	0°
329	N4.47266	E101.03793	10:38:23	134	0	00:00:01	0	0°
330	N4.47266	E101.03793	10:38:24	134	0	00:00:01	0	0°
331	N4.47266	E101.03793	10:38:25	134	0	00:00:01	0	0°
332	N4.47266	E101.03793	10:38:26	133	0	00:00:01	0	0°
333	N4.47266	E101.03793	10:38:27	133	0	00:00:01	0	0°
334	N4.47266	E101.03793	10:38:28	133	0	00:00:01	0	0°
335	N4.47268	E101.03791	10:38:29	133	11	00:00:01	7.5	315°
336	N4.47270	E101.03791	10:38:30	133	8	00:00:01	5.3	0°
337	N4.47274	E101.03791	10:38:31	133	16	00:00:01	11	0°
338	N4.47277	E101.03793	10:38:32	131	11	00:00:01	7.5	45°
339	N4.47281	E101.03796	10:38:33	131	18	00:00:01	12	26°
340	N4.47283	E101.03798	10:38:34	131	11	00:00:01	7.5	45°
341	N4.47287	E101.03800	10:38:35	134	18	00:00:01	12	26°
342	N4.47292	E101.03802	10:38:36	133	18	00:00:01	12	26°
343	N4.47296	E101.03806	10:38:37	134	22	00:00:01	15	45°
344	N4.47300	E101.03811	10:38:38	134	22	00:00:01	15	45°
345	N4.47307	E101.03817	10:38:39	134	33	00:00:01	23	45°
346	N4.47311	E101.03823	10:38:40	134	28	00:00:01	19	56°
347	N4.47315	E101.03828	10:38:41	133	22	00:00:01	15	45°
348	N4.47322	E101.03834	10:38:42	131	33	00:00:01	23	45°
349	N4.47326	E101.03841	10:38:43	131	28	00:00:01	19	56°

350	N4.47330	E101.03847	10:38:44	130	28	00:00:01	19	56°
351	N4.47332	E101.03853	10:38:45	130	25	00:00:01	17	72°
352	N4.47337	E101.03862	10:38:46	130	35	00:00:01	24	63°
353	N4.47339	E101.03871	10:38:47	130	32	00:00:01	22	76°
354	N4.47337	E101.03879	10:38:48	131	32	00:00:01	22	104°
355	N4.47337	E101.03888	10:38:49	131	31	00:00:01	21	90°
356	N4.47337	E101.03896	10:38:50	131	31	00:00:01	21	90°
357	N4.47335	E101.03905	10:38:51	131	32	00:00:01	22	104°
358	N4.47332	E101.03914	10:38:52	131	32	00:00:01	22	104°
359	N4.47330	E101.03922	10:38:53	131	32	00:00:01	22	104°
360	N4.47326	E101.03931	10:38:54	133	35	00:00:01	24	117°
361	N4.47324	E101.03937	10:38:55	131	25	00:00:01	17	108°
362	N4.47320	E101.03946	10:38:56	131	35	00:00:01	24	117°
363	N4.47315	E101.03954	10:38:57	131	35	00:00:01	24	117°
364	N4.47313	E101.03963	10:38:58	130	32	00:00:01	22	104°
365	N4.47309	E101.03971	10:38:59	128	35	00:00:01	24	117°
366	N4.47302	E101.03978	10:39:00	128	33	00:00:01	23	135°
367	N4.47298	E101.03987	10:39:01	127	35	00:00:01	24	117°
368	N4.47294	E101.03995	10:39:02	127	35	00:00:01	24	117°
369	N4.47289	E101.04004	10:39:03	125	35	00:00:01	24	117°
370	N4.47285	E101.04012	10:39:04	127	35	00:00:01	24	117°
371	N4.47279	E101.04021	10:39:05	127	39	00:00:01	27	127°
372	N4.47274	E101.04029	10:39:06	127	35	00:00:01	24	117°
373	N4.47270	E101.04036	10:39:07	127	28	00:00:01	19	124°
374	N4.47266	E101.04044	10:39:08	128	35	00:00:01	24	117°
375	N4.47264	E101.04051	10:39:09	130	25	00:00:01	17	108°
376	N4.47259	E101.04057	10:39:10	130	28	00:00:01	19	124°
377	N4.47255	E101.04064	10:39:11	130	28	00:00:01	19	124°
378	N4.47253	E101.04070	10:39:12	130	25	00:00:01	17	108°
379	N4.47251	E101.04077	10:39:13	130	25	00:00:01	17	108°
380	N4.47249	E101.04081	10:39:14	130	17	00:00:01	12	117°
381	N4.47247	E101.04087	10:39:15	130	25	00:00:01	17	108°
382	N4.47244	E101.04094	10:39:16	130	25	00:00:01	17	108°
383	N4.47242	E101.04096	10:39:17	130	11	00:00:01	7.5	135°
384	N4.47242	E101.04098	10:39:18	133	8	00:00:01	5.3	90°
385	N4.47242	E101.04098	10:39:19	134	0	00:00:01	0	0°
386	N4.47242	E101.04098	10:39:20	134	0	00:00:01	0	0°

387	N4.47240	E101.04098	10:39:21	134	8	00:00:01	5.3	180°
388	N4.47240	E101.04098	10:39:22	134	0	00:00:01	0	0°
389	N4.47240	E101.04098	10:39:23	134	0	00:00:01	0	0°
390	N4.47240	E101.04098	10:39:24	134	0	00:00:01	0	0°
391	N4.47240	E101.04098	10:39:25	136	0	00:00:01	0	0°
392	N4.47238	E101.04098	10:39:26	136	8	00:00:01	5.3	180°
393	N4.47238	E101.04098	10:39:27	136	0	00:00:01	0	0°
394	N4.47238	E101.04098	10:39:28	134	0	00:00:01	0	0°
395	N4.47238	E101.04098	10:39:29	134	0	00:00:01	0	0°
396	N4.47238	E101.04098	10:39:30	134	0	00:00:01	0	0°
397	N4.47238	E101.04098	10:39:31	133	0	00:00:01	0	0°
398	N4.47238	E101.04098	10:39:32	133	0	00:00:01	0	0°
399	N4.47238	E101.04098	10:39:33	136	0	00:00:01	0	0°
400	N4.47238	E101.04098	10:39:34	136	0	00:00:01	0	0°
401	N4.47238	E101.04098	10:39:35	136	0	00:00:01	0	0°
402	N4.47238	E101.04100	10:39:36	138	8	00:00:01	5.3	90°
403	N4.47238	E101.04102	10:39:37	136	8	00:00:01	5.3	90°
404	N4.47236	E101.04105	10:39:38	136	11	00:00:01	7.5	135°
405	N4.47236	E101.04107	10:39:39	136	8	00:00:01	5.3	90°
406	N4.47236	E101.04111	10:39:40	134	16	00:00:01	11	90°
407	N4.47238	E101.04115	10:39:41	134	17	00:00:01	12	63°
408	N4.47238	E101.04120	10:39:42	133	16	00:00:01	11	90°
409	N4.47238	E101.04126	10:39:43	131	23	00:00:01	16	90°
410	N4.47240	E101.04130	10:39:44	130	17	00:00:01	12	63°
411	N4.47240	E101.04137	10:39:45	130	23	00:00:01	16	90°
412	N4.47240	E101.04141	10:39:46	130	16	00:00:01	11	90°
413	N4.47242	E101.04147	10:39:47	131	25	00:00:01	17	72°
414	N4.47242	E101.04154	10:39:48	131	23	00:00:01	16	90°
415	N4.47242	E101.04160	10:39:49	131	23	00:00:01	16	90°
416	N4.47244	E101.04165	10:39:50	133	17	00:00:01	12	63°
417	N4.47244	E101.04171	10:39:51	133	23	00:00:01	16	90°
418	N4.47244	E101.04175	10:39:52	134	16	00:00:01	11	90°
419	N4.47247	E101.04182	10:39:53	136	25	00:00:01	17	72°
420	N4.47247	E101.04186	10:39:54	136	16	00:00:01	11	90°
421	N4.47247	E101.04190	10:39:55	136	16	00:00:01	11	90°
422	N4.47247	E101.04195	10:39:56	136	16	00:00:01	11	90°
423	N4.47247	E101.04201	10:39:57	134	23	00:00:01	16	90°

424	N4.47247	E101.04203	10:39:58	134	8	00:00:01	5.3	90°
425	N4.47247	E101.04208	10:39:59	133	16	00:00:01	11	90°
426	N4.47247	E101.04212	10:40:00	131	16	00:00:01	11	90°
427	N4.47244	E101.04218	10:40:02	131	25	00:00:02	8.4	108°
428	N4.47238	E101.04220	10:40:03	131	25	00:00:01	17	162°
429	N4.47232	E101.04220	10:40:04	130	24	00:00:01	16	180°
430	N4.47225	E101.04223	10:40:05	128	25	00:00:01	17	162°
431	N4.47219	E101.04223	10:40:06	127	24	00:00:01	16	180°
432	N4.47210	E101.04225	10:40:07	125	32	00:00:01	22	166°
433	N4.47204	E101.04227	10:40:08	125	25	00:00:01	17	162°
434	N4.47195	E101.04229	10:40:09	125	32	00:00:01	22	166°
435	N4.47186	E101.04229	10:40:10	123	31	00:00:01	21	180°
436	N4.47180	E101.04231	10:40:11	123	25	00:00:01	17	162°
437	N4.47171	E101.04233	10:40:12	123	32	00:00:01	22	166°
438	N4.47165	E101.04235	10:40:13	123	25	00:00:01	17	162°
439	N4.47156	E101.04238	10:40:14	122	32	00:00:01	22	166°
440	N4.47150	E101.04240	10:40:15	122	25	00:00:01	17	162°
441	N4.47144	E101.04240	10:40:16	120	24	00:00:01	16	180°
442	N4.47137	E101.04242	10:40:17	119	25	00:00:01	17	162°
443	N4.47131	E101.04244	10:40:18	119	25	00:00:01	17	162°
444	N4.47126	E101.04244	10:40:19	119	16	00:00:01	11	180°
445	N4.47122	E101.04246	10:40:20	117	18	00:00:01	12	154°
446	N4.47118	E101.04244	10:40:21	117	18	00:00:01	12	206°
447	N4.47116	E101.04244	10:40:22	117	8	00:00:01	5.3	180°
448	N4.47114	E101.04242	10:40:23	116	11	00:00:01	7.5	225°
449	N4.47114	E101.04242	10:40:24	117	0	00:00:01	0	0°
450	N4.47114	E101.04242	10:40:25	117	0	00:00:01	0	0°
451	N4.47111	E101.04242	10:40:26	116	8	00:00:01	5.3	180°
452	N4.47111	E101.04242	10:40:27	116	0	00:00:01	0	0°
453	N4.47111	E101.04240	10:40:28	116	8	00:00:01	5.3	270°
454	N4.47111	E101.04240	10:40:29	116	0	00:00:01	0	0°
455	N4.47111	E101.04240	10:40:30	116	0	00:00:01	0	0°
456	N4.47111	E101.04240	10:40:31	116	0	00:00:01	0	0°
457	N4.47111	E101.04240	10:40:32	116	0	00:00:01	0	0°
458	N4.47111	E101.04240	10:40:33	116	0	00:00:01	0	0°
459	N4.47111	E101.04240	10:40:34	116	0	00:00:01	0	0°
460	N4.47111	E101.04240	10:40:35	116	0	00:00:01	0	0°

461	N4.47111	E101.04240	10:40:36	116	0	00:00:01	0	0°
462	N4.47111	E101.04240	10:40:37	117	0	00:00:01	0	0°
463	N4.47111	E101.04240	10:40:38	117	0	00:00:01	0	0°
464	N4.47111	E101.04240	10:40:39	117	0	00:00:01	0	0°
465	N4.47111	E101.04240	10:40:40	117	0	00:00:01	0	0°
466	N4.47111	E101.04240	10:40:41	117	0	00:00:01	0	0°
467	N4.47111	E101.04240	10:40:42	117	0	00:00:01	0	0°
468	N4.47111	E101.04240	10:40:43	117	0	00:00:01	0	0°
469	N4.47111	E101.04238	10:40:44	117	8	00:00:01	5.3	270°
470	N4.47111	E101.04238	10:40:45	117	0	00:00:01	0	0°
471	N4.47111	E101.04238	10:40:46	117	0	00:00:01	0	0°
472	N4.47111	E101.04238	10:40:47	117	0	00:00:01	0	0°
473	N4.47109	E101.04235	10:40:48	116	11	00:00:01	7.5	225°
474	N4.47109	E101.04233	10:40:49	114	8	00:00:01	5.3	270°
475	N4.47109	E101.04231	10:40:50	114	8	00:00:01	5.3	270°
476	N4.47111	E101.04229	10:40:51	114	11	00:00:01	7.5	315°
477	N4.47114	E101.04225	10:40:52	114	17	00:00:01	12	297°
478	N4.47118	E101.04223	10:40:53	116	18	00:00:01	12	334°
479	N4.47120	E101.04220	10:40:54	116	11	00:00:01	7.5	315°
480	N4.47124	E101.04218	10:40:55	116	18	00:00:01	12	334°
481	N4.47129	E101.04214	10:40:56	114	22	00:00:01	15	315°
482	N4.47133	E101.04210	10:40:57	114	22	00:00:01	15	315°
483	N4.47139	E101.04205	10:40:58	114	28	00:00:01	19	326°
484	N4.47144	E101.04201	10:40:59	114	22	00:00:01	15	315°
485	N4.47148	E101.04197	10:41:00	114	22	00:00:01	15	315°
486	N4.47154	E101.04192	10:41:01	112	28	00:00:01	19	326°
487	N4.47159	E101.04188	10:41:02	112	22	00:00:01	15	315°
488	N4.47161	E101.04182	10:41:03	122	25	00:00:01	17	288°
489	N4.47167	E101.04184	10:41:04	125	25	00:00:01	17	18°
490	N4.47174	E101.04180	10:41:05	127	28	00:00:01	19	326°
491	N4.47184	E101.04171	10:41:07	125	50	00:00:02	17	321°
492	N4.47189	E101.04167	10:41:08	123	22	00:00:01	15	315°
493	N4.47193	E101.04158	10:41:09	116	35	00:00:01	24	297°
494	N4.47199	E101.04156	10:41:10	116	25	00:00:01	17	342°
495	N4.47204	E101.04152	10:41:11	117	22	00:00:01	15	315°
496	N4.47208	E101.04147	10:41:12	117	22	00:00:01	15	315°
497	N4.47212	E101.04143	10:41:13	117	22	00:00:01	15	315°

498	N4.47219	E101.04139	10:41:14	117	28	00:00:01	19	326°
499	N4.47223	E101.04139	10:41:15	116	16	00:00:01	11	0°
500	N4.47227	E101.04137	10:41:16	117	18	00:00:01	12	334°
501	N4.47232	E101.04139	10:41:17	117	18	00:00:01	12	26°
502	N4.47236	E101.04141	10:41:18	117	18	00:00:01	12	26°
503	N4.47238	E101.04145	10:41:19	117	17	00:00:01	12	63°
504	N4.47238	E101.04150	10:41:20	117	16	00:00:01	11	90°
505	N4.47238	E101.04156	10:41:21	117	23	00:00:01	16	90°
506	N4.47238	E101.04162	10:41:22	119	23	00:00:01	16	90°
507	N4.47238	E101.04169	10:41:23	120	23	00:00:01	16	90°
508	N4.47238	E101.04173	10:41:24	120	16	00:00:01	11	90°
509	N4.47240	E101.04182	10:41:25	122	32	00:00:01	22	76°
510	N4.47240	E101.04188	10:41:26	123	23	00:00:01	16	90°
511	N4.47240	E101.04195	10:41:27	125	23	00:00:01	16	90°
512	N4.47242	E101.04203	10:41:28	127	32	00:00:01	22	76°
513	N4.47242	E101.04210	10:41:29	128	23	00:00:01	16	90°
514	N4.47242	E101.04218	10:41:30	130	31	00:00:01	21	90°
515	N4.47244	E101.04227	10:41:31	130	32	00:00:01	22	76°
516	N4.47244	E101.04235	10:41:32	131	31	00:00:01	21	90°
517	N4.47247	E101.04244	10:41:33	131	32	00:00:01	22	76°
518	N4.47247	E101.04253	10:41:34	131	31	00:00:01	21	90°
519	N4.47249	E101.04259	10:41:35	131	25	00:00:01	17	72°
520	N4.47249	E101.04268	10:41:36	131	31	00:00:01	21	90°
521	N4.47249	E101.04276	10:41:37	131	31	00:00:01	21	90°
522	N4.47251	E101.04287	10:41:38	131	40	00:00:01	27	79°
523	N4.47251	E101.04295	10:41:39	130	31	00:00:01	21	90°
524	N4.47253	E101.04304	10:41:40	128	32	00:00:01	22	76°
525	N4.47253	E101.04315	10:41:41	128	39	00:00:01	27	90°
526	N4.47255	E101.04326	10:41:42	127	40	00:00:01	27	79°
527	N4.47255	E101.04334	10:41:43	125	31	00:00:01	21	90°
528	N4.47257	E101.04345	10:41:44	123	40	00:00:01	27	79°
529	N4.47259	E101.04353	10:41:45	123	32	00:00:01	22	76°
530	N4.47262	E101.04364	10:41:46	120	40	00:00:01	27	79°
531	N4.47266	E101.04375	10:41:47	117	42	00:00:01	29	68°
532	N4.47270	E101.04383	10:41:48	114	35	00:00:01	24	63°
533	N4.47277	E101.04392	10:41:49	112	39	00:00:01	27	53°
534	N4.47285	E101.04398	10:41:50	109	39	00:00:01	27	37°

535	N4.47292	E101.04407	10:41:51	108	39	00:00:01	27	53°
536	N4.47300	E101.04414	10:41:52	106	39	00:00:01	27	37°
537	N4.47309	E101.04422	10:41:53	105	44	00:00:01	30	45°
538	N4.47317	E101.04429	10:41:54	105	39	00:00:01	27	37°
539	N4.47324	E101.04437	10:41:55	105	39	00:00:01	27	53°
540	N4.47330	E101.04444	10:41:56	105	33	00:00:01	23	45°
541	N4.47337	E101.04452	10:41:57	103	39	00:00:01	27	53°
542	N4.47343	E101.04461	10:41:58	103	39	00:00:01	27	53°
543	N4.47347	E101.04469	10:41:59	103	35	00:00:01	24	63°
544	N4.47354	E101.04478	10:42:00	101	39	00:00:01	27	53°
545	N4.47356	E101.04484	10:42:01	101	25	00:00:01	17	72°
546	N4.47360	E101.04493	10:42:02	103	35	00:00:01	24	63°
547	N4.47365	E101.04501	10:42:03	103	35	00:00:01	24	63°
548	N4.47369	E101.04510	10:42:04	101	35	00:00:01	24	63°
549	N4.47371	E101.04519	10:42:05	103	32	00:00:01	22	76°
550	N4.47375	E101.04527	10:42:06	103	35	00:00:01	24	63°
551	N4.47377	E101.04536	10:42:07	103	32	00:00:01	22	76°
552	N4.47382	E101.04547	10:42:08	101	42	00:00:01	29	68°
553	N4.47386	E101.04555	10:42:09	101	35	00:00:01	24	63°
554	N4.47390	E101.04566	10:42:10	101	42	00:00:01	29	68°
555	N4.47392	E101.04574	10:42:11	101	32	00:00:01	22	76°
556	N4.47397	E101.04585	10:42:12	101	42	00:00:01	29	68°
557	N4.47401	E101.04596	10:42:13	103	42	00:00:01	29	68°
558	N4.47405	E101.04607	10:42:14	105	42	00:00:01	29	68°
559	N4.47410	E101.04617	10:42:15	105	42	00:00:01	29	68°
560	N4.47414	E101.04626	10:42:16	105	35	00:00:01	24	63°
561	N4.47416	E101.04637	10:42:17	105	40	00:00:01	27	79°
562	N4.47420	E101.04647	10:42:18	106	42	00:00:01	29	68°

Table A.1: Relevant Raw Trajectory

Appendix B

ViTracker Tutorial

B.1 Introduction

This user manual is for ViTracker, vehicle tracking simulator platform. You can download the executable from *http://dewandaru.googlepages.com/source.zip* ViTracker allows the vehicle tracking or map-matching algorithms implemented and tested with various scenarios. ViTracker is equipped with GPS simulator to give some position signal randomness, if needed. Alternatively, the vehicle track could be edited, or be read from the MapSource(tm) output text file. ViTracker is also able to produce a visualization of the map-matching process in real-time.

B.2 Basic Usage

The first time the application is launched (by running prjViTracker.Exe), the user will be presented with the blank map. What you need to prepare is the map and the simulation file. The map file is a simple text initialization file (INI file). You can prepare the file like the example below, or just open the map file that is supplied with the example (named **map.ini**). To open the example, choose menu File/Open Map and browse for the map.ini that is on the folder BatuGajah (the example folder).

;Note that ';' start a comment line and thus the line will be ignored by the system
;0=(fill with your own BMP)
;(name for layer 1)=(Ini file name for the layer 1)
;(name for layer 2)=(Ini file name for the layer 2)
;etc.
[layers]

0=Batugajah.bmp
RoadNet=layer1.ini
Vehicle Trajectory=VTrack.ini

After successfully loading the map file, you will be presented with the aerial view of Batu Gajah town, which is taken from Google Earth. You could supply any bitmaps that you like, just update the map file, at entry layer 0, to point to your custom bitmap.

Next, open and browse the simulation file (Sim.Ini) from the menu Simulation/Open. This will load the necessary simulation information. If you want to load a recorded true gps points, then the simulation file should have this key/value pair under [config] section:

track=true_gps,Vehicle Trajectory

The "track" key is necessary. The true_gps could be replaced with simulated_gps, if you like to have a simulated one. The next value ("Vehicle Trajectory") must refer to the name of one existing layer defined in the [layers] section in the map file. In this case, we would like to appoint the Vehicle Trajectory layer to serve as the GPS points record. The Vehicle Trajectory layer is normally prepared by a special command Simulation/Import Garmin Tracks.

After you have open both map and simulation file, you can press the "Run" to see how the active map-matching algorithm performed. The vehicle and its short trace is represented by a small green arrow. The map-matched route is represented by a thick blue line. The overall (future and past) route for the true_gps option is represented by a circular red line.

B.3 Digitizing a Map

To prepare map, you can start from the bitmap as the background, then by encoding the important nodes and arcs based on that bitmap. To start, get a background bitmap that you want to digitize. It should be aerial with some easily identifiable landmarks. The landmarks will be useful in the georeferencing process later on. Assume that the background bitmap is named "map.bmp", and placed in the same folder as the map file "map.ini", folder F. Put "map.bmp" within the [layers] section of "map.ini", specifically, in the special layer 0. Then, you can open and modify the "layer1.ini" that is available in the example provided. Save the modification in folder F. Also, add within the [layers] section, a reference to "layer1.ini".

The next step is to add relevant nodes within the blank map and connecting them. The nodes are placed in the road intersection, or in a relevant road curvature. Please pick the "Editor" tab in the far right. You create nodes by picking Add node command and then clicking on the screen. Then you create arcs by selecting the nodes and then issue a Create Road from Nodes. Others are supportive commands and can be explored easily, especially noting the instruction messages that appear on the center of the application window.

After this process, it is necessary to attach / register the map to the real coordinate. This is done by selecting the "Set Registration" command. You will be asked with a reference node Id (the number that is shown on the node) and a location (Lat/Long) of that node. This is done two times and you must enter in a string formatted here refnode:xabs:yabs:refnode2:xabs2:yabs2. for example,
1:101.041361:4.476494:49:101.037825:4.472722