UNIVERSITY OF CALIFORNIA
RIVERSIDE

GIS for Mapping of Lane-Level Data and Re-Creation in Real Time for Navigation

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Electrical Engineering

by

Behlul Zoeb Sutarwala

March 2011

Thesis Committee:

Dr. Jay A. Farrell, Chairperson
Dr. Mathew Barth
Dr. Shrikanth Krishnamurthy

The Thesis of Behlul Zoeb Sutarwala is approved.

Committee Chairperson

University of California, Riverside

ACKNOWLEDGEMENTS

I would like to thank Professor Jay Farrell for his mentorship without which this thesis would not have been possible. I would also like to thank Professor Matthew Barth and Professor Shrikanth Krishnamurthy for serving on my committee.

I would like to thank my fellow graduate students Arvind Ramanandan and Anning Chen for their support and help in data collection and testing.

I would also like to thank Akshay Morye for his help in database management using ArcGIS.

Lastly, but most importantly, I would like to thank my parents, Nafisa and Zoeb Sutarwala, my brother Quresh and the rest of my family for their support and patience.

ABSTRACT OF THE THESIS

GIS for Mapping of Lane-Level Data and Re-Creation in Real Time for Navigation

by

Behlul Zoeb Sutarwala

Master of Science, Graduate Program in Electrical Engineering
University of California, Riverside, March 2011
Dr. Jay A. Farrell, Chairperson

Advanced navigation systems for advanced driver assistance systems for safety of vehicle occupants and for autonomous vehicles require high accuracy digital maps. These maps should contain enough attributes and precision to be able to guide the vehicles within their lanes. Along with these digital maps we also require them to process the data in real time. In this Thesis we will design a geodatabase for such a lane-level digital map using a nodal approach. We collect decimeter accuracy data by using a datalog vehicle (Rover). We followed GIS practices for database development and use GIS management tools such as ArcGIS. In the second half of the thesis we implemented a Human Machine Interface for an Advanced Drive Assistance System. This system will query data from the geodatabase and process it to graphically display on screen the vehicle and the lane-level map of the region around it. It will also display a predicted vehicle position to

notify drivers of future lane departures. We tested this system along the Interstate 80 in the Donner Lake Region where it was a part of a Snow Plow Guidance Project implemented by UCR for CALTRANS.

Contents

List of Tables

List of Figures

Chapter One: Introduction

1.1 Background

Navigation systems based on Global Positioning System (GPS) are commercially available and can pin-point our position with an accuracy of approximately 20 meters. These are used in vehicles to help plan a route and guide the driver to the desired destination.

Advanced applications in vehicle navigation systems to assist drivers have been undertaken by the Transit Vehicle and Automation (VAA) program. These include lane departure warning systems [9], [16], [18] which reduce the chance of accidents, the intelligent cruise control [19] and lane-level driver guidance [12] which helps reduce excessive and last minute lane changes.

Vision systems have been used with GPS receivers to develop a lane departure warning system [21] but these fail if the lane markers are not visible.

We intend to further improve these automotive navigation systems so that we increase safety of the vehicle occupants.

1.2 Motivation

The motivation for this project was to improve vehicle occupant safety. If we can convey to the driver of the motor vehicle, the current vehicle position and the predicted vehicle path for the next few seconds, he would have more time to correct himself if required in order to avoid an accident. For example, it would be extremely helpful to Snow Plow drivers who have little or no visual confirmation of road or lane edges. Such an Advanced Driver Assistance System (ADAS) could also be used as an early warning system for lane departure.

1.3 Aim

We want to design an Advanced Driver Assistance System which would convey the current vehicle position and the predicted path to the driver. To do so, we need to determine the current vehicle state [position, attitude, speed] and the predicted vehicle path and then display this information along with a lane-level map of region surrounding the vehicle.

To find the current vehicle state, we use a GPS receiver with carrier phase differential corrections and an Inertial Measurement Unit [1]. From this we calculate the predicted vehicle path.

To display the map around the vehicle we need a digital map containing lane-level data. This map, when queried should be able to supply information in real time. We also need to be able to process data from the maps to display in real time and hence the need for the database to be concise.

Current maps do not contain enough lane-level data to be able to locate the position of the vehicle in the lane due to the unavailability of decimeter accuracy positional data [26]. Since, we have a system to collect positional data with centimeter level accuracy and create maps with decimeter accuracy [3], we use this data to create our own maps. The approach to database design and tools used is discussed in detail in Chapter 2.

Once we have built the digital map of the desired roadway network, we need to query this database in real time and convey to the driver this information along with the predicted vehicle path. This would enable the driver to take corrective steps in order to avoid an accident. This step is discussed in Chapter 3.

Figure 1 shows the general workflow of our implementation. It shows the offline section, non-real time which is the building of the lane-level maps with decimeter accuracy and the online section, real time system to access maps and display information in real time.

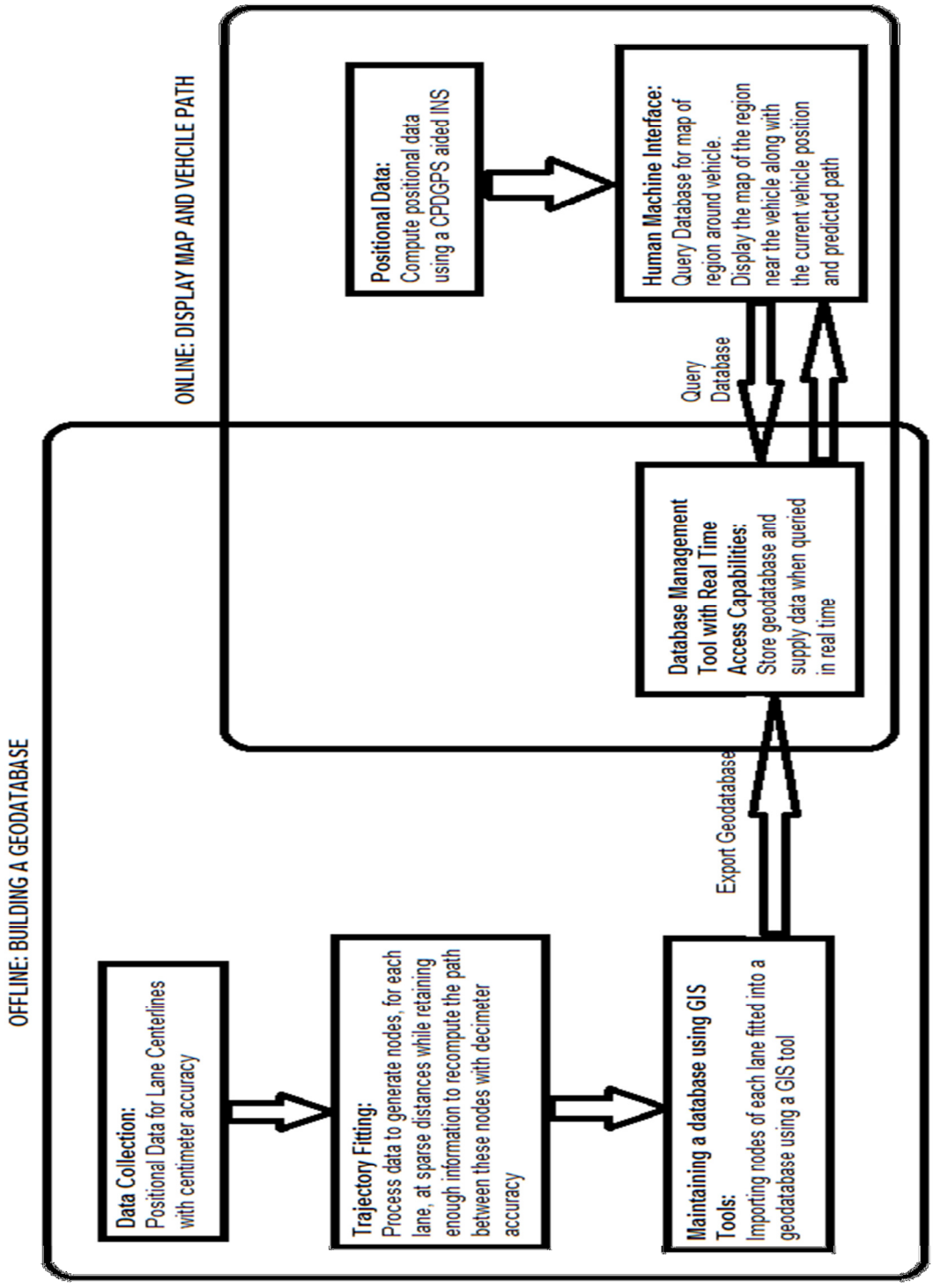


Figure 1: Implementation Flow of our ADAS.

Chapter Two: GIS for Mapping Lane-level Data

2.1 GIS for Transportation.

Geographical Information Systems (GIS) are systems which are used to capture, analyze, interpret and manage data related to their location. It is the merging of cartography, statistical analysis, and database technology. In general a GIS system creates a database with the positional data of a certain area and some statistical attributes in that area. For example it may store, the Latitude, Longitude, Altitude (positional data) for the center of each city and the daily temperatures corresponding to them. Such a database created with reference the geographical position is known as a geodatabase.

GIS is widely used in the design of transportation systems like road, rail and water networks. This branch of GIS is commonly known as GIS-T i.e. GIS for Transportation.

2.2 GIS Management Software

We require two main functionalities in our decimeter accuracy lane-level maps:

1. A scalable database with tools to maintain the maps.

2. Ability to retrieve data within a small region in real time.

To build and maintain our maps we had a few choices of available GIS tools. Table 1 provides a selected list of such tools we considered to build our digital maps.

GIS Management Software	Features
TransCAD	• It is an application for public transport applications. • In addition to point, line, polygon and raster image layers in a GIS map, it also supports route system layers. • It possesses tools for creating, manipulating and displaying routes. • TransCAD uses network data structure to support routing and network optimization models. • Trip generation, mode choice and traffic assignment models that support transportation planning and travel forecasting are supported. • It also has a set of dynamic segmentation and linear referencing tools for managing highways and other networks.
ArcGIS	• It is considered to have the most amount functionality when it comes to use in Transportation applications. • It comes with software extensions like Network Analyst, which can be used to make a network of trajectories. • It can store vector and raster data in shape files. It can also store data in a proprietary relational database management system (RDBMS) format called Geodatabase. • It has the capability to export the geodatabase to PostGRE SQL, MS SQL Server.

GRASS	• It is a free, open source GIS, capable of handling vector (topological), raster, image processing and graphic data. • It can be used on Mac OS, Windows and Linux platforms. • It was released under the GNU-GPL license. • The software is interfaced through a GUI using an internal GUI or by plugging into GRASS via Quantum GIS, which is available for free. • The latest release of the software introduced new topological 2-D and 3-D vector engine and support for vector network analysis. • Attributes for the features are managed in a '.dbf' file or an SQL-based DBMS such as MySQL, Postgre SQL / PostGIS and SQLite.
Microsoft MapPoint	• It is a program created by Microsoft to allow users to view, edit and integrate maps. • Visualization and analysis of the custom data is possible. • Many of Microsoft's acquisitions have supplemented data and feature integration. • It includes all the functionality of the most recent version of streets and trips i.e. a consumer mapping software. • This software also integrates with other Microsoft products like MS Office and a VBA interface, allowing automation of MapPoint environment. • The latest edition of the software has GPS integration features. • This software is essentially aimed at business users.

Table 1: A few GIS Management Softwares and their features.[29]

After our initial review of the capabilities of each of the above software we realized none of them would be able to deliver data in real time for a query from our display program. Hence we decided to use one of these tools to build our geodatabase and to export it to another database management tool which can be queried in real time. Some of these are listed in Table 2.

Database Management Software	Cost / Features
SQL Server	Free : Express Edition – Database Size Limit 1GB Paid : Enterprise or Personal Edition – No size limit, Product Support
Postgre SQL	Free : Open source, RDBMS
MySQL	Free : Open source, RDBMS

Table 2: Database Management Software

The database management software listed in Table 2, all support spatial databases, i.e. they are optimized to store and query objects in space for example points, lines polygons, etc.

After evaluating the functionalities of the GIS Management Software in Table 1 and their compatibility with Database Management Software in Table 2, we chose ArcGIS, for the following reasons:

1. It provided extensive tools to insert features into the geodatabase and to maintain roadway networks.
2. It had the capability to export the completed geodatabase to Microsoft SQL Server.
3. A student license version of ArcInfo, an ArcGIS package with extensions that supports the above was available to us for a low cost of approximately $200.

2.3 Preparation of Geodatabase

We will use the "Nodal Approach" as described in [4] to represent the roadway network in our geodatabase. This is done in three steps.

1. Acquire data representing lane centerlines with centimeter accuracy.
2. Fit the data to a trajectory depicting the lane centerlines with decimeter accuracy.
3. Store the data in a manner that:
    - It represents each lane on a road
    - We can reproduce the lane trajectory from this data with decimeter accuracy.
    - We can query this data in real time.

Steps 1 and 2 are part of a separate project being carried out under Prof. Farrell. In this thesis, we will describe in detail Step 3.

2.3.1 Data Collection

We acquire data using a Carrier Phase Differential GPS (CPDGPS) aided INS system [1][11]. This system is referred to as the Rover is mounted on a vehicle. The vehicle is driven along a lane, with best efforts to keep the vehicle on the center of that lane. The carrier phase differential corrections are transmitted by the Base system. Both the Base and Rover systems run on software implemented in C++ on a computer running the Ubuntu flavor of Linux. Even though the data collected using the CPDGPS system has

centimeter level accuracy, human error is introduced due to the driver trying to keep the vehicle on the center of the lane. This data is collected at 15Hz.

Both the base and rover software were developed at the Controls and Robotics Lab at UCR under the guidance of Prof. Jay Farrell.

2.3.2 Trajectory Fitting

The data acquired in Section 2.3.1 is dense positional data, which is not uniformly distributed as it depends on the speed of the vehicle. The slower the vehicle, the denser our data is. To efficiently store this lane centerline data, we need to store points at sparser intervals which can represent this data with decimeter accuracy. We fit this data to a trajectory depicting the lane centerlines. We use the trajectory fitting code developed by Prof. Farrell in MATLAB. This takes place in two steps. First we plot all the positional data to visually check the integrity of the raw data. Then we cross reference this data with the timings we wrote down during data collection and parse the data into different sections for each lane. We now plot this again on screen one lane or one section of a lane at a time and remove the points which seem to be obviously corrupted. This is done by using the function 'parse_data4traj.m'.

Then we use this data to fit a Hermite spline through it using least squares to generate nodes for each lane using the 'traj_fit.m' function. This is explained in detail in [3][4].

The data for each lane is stored as a sequence of nodes with each node having different attributes to describe the lane.

2.3.3 Node Attributes

Each node has certain attributes required to recreate the lanes with decimeter accuracy. The table below shows a few sample nodes and attributes which we used to create the geodatabase.

Node No.	Latitude (Deg.)	Longitude (Deg.)	Altitude (m)	Yaw (rad)	Grade	Curvature	Arclength (m)
1	D:39 M:20 S:20.5938	D:-120 M:20 S:55.035	2177.36	1.207	0.023	0.0001	1.00
2	D:39 M:20 S:21.7752	D:-120 M:20 S:50.949	2177.76	1.217	-0.006	0.0001	105.39
3	D:39 M:20 S:22.9446	D:-120 M:20 S:46.86	2178.91	1.207	-0.012	0.0000	209.78

Table 3: Table showing the nodes and node attributes for a lane with 3 nodes (Latitude and Longitude are stored in decimal degrees. They are listed as DMS for representation purposes)

The first three parameters i.e. latitude, longitude and altitude give us the positional data in the geodetic ECEF plane. Even though we fit the data in the tangent plane we transform it to the ECEF frame before we build the database. This allows us to use the same database for different base locations.

'Yaw' is defined by the angle between the lane trajectory at that point and the true north and is represented in degrees.

'Grade' is defined as the inclination of the road with reference to the local tangent plane.

'Curvature' is defined as the rate of change of the unit tangent vector at that point.

'Arclength' is the distance travelled along the lane trajectory from the beginning of this section of the lane to the current node.

The sequence of nodes for each lane is stored in a comma separated value ('.csv') file.

For verification purposes we also create Keyhole Markup Language (KML) files which are XML-based files for expressing geographic data. They are used to display geographic data in any Earth browsers such as Google Earth. We generate two types of KML files, one that contains nodes that will be inserted into the geodatabase and one which contains a dense set of points generated between each of the nodes using a Hermite polynomial[3][4].

2.3.4 Verification using Google Earth

Now that we had a list of nodes for each lane along with their attributes, we projected these in Google Earth by importing the '.kml' created above with the same nodes. We made sure that the lanes and roads were consistent with the background on Google Earth. A sample projection of few lanes is shown in Figure 2(a) and 2(b). In these figures, the yellow place marks depict the position of the nodes and the red line is the generated trajectory using the node attributes.

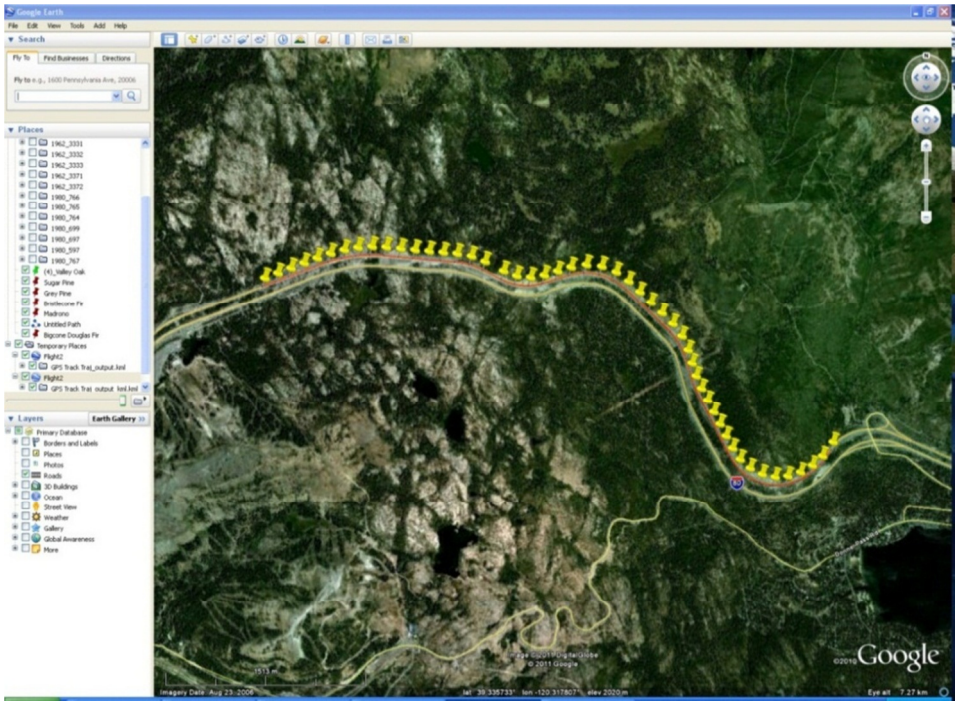


Figure 2(a)

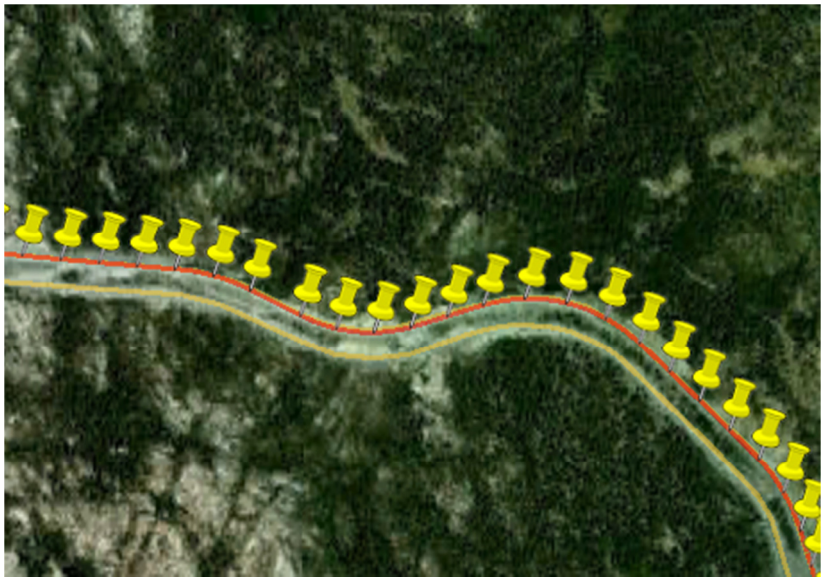


Figure 2 (b)

Figure 2: KML files generated using the nodes and the calculated points using the Hermite Spline.

2.3.5 Database Design

Until now we have processed data for a single lane segment and stored it as a list of nodes. Now, we need to include this as a part of a roadway network and store the data from all lanes in a database that can be queries in real time. As discussed in section 2.2, we used ArcGIS 9.3 as our GIS database management tool and MS SQL Server as for real time database access. We created a map in ArcMap by importing nodes and their attributes for each lane generated in section 2.3.2. Figure 3 shows a map created with two lanes.

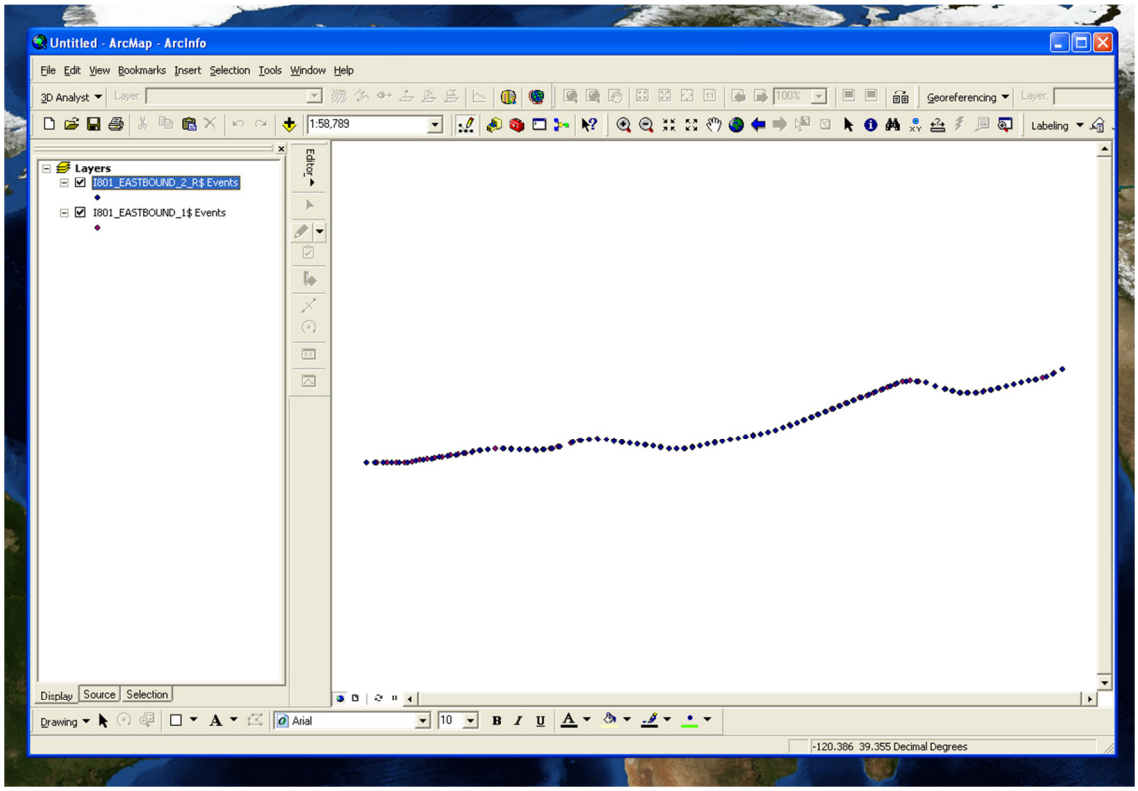


Figure 3: Map created using ArcGIS.

Now we used ArcCatalog to create a geodatabase. After which we inserted all the lanes on our map into the geodatabase.

It consisted of a separate table for each lane stored. There were numerous other tables created. We only used one other table called 'SDE_layers' which contained spatial information like the maximum and minimum values of the latitude and longitude of each lane.

We differentiated our roads and lanes by the nomenclature of each of these lanes. Our lanes and thus tables in the geodatabase were named in the format

ABC_DIR_123_R

where, 'ABC' represents the name of the road e.g. 'I80'; 'DIR' represents the direction in which the road is heading e.g. 'EASTBOUND'; '123' represents the lane number starting from the left most lane as '1' and 'R' is an addition only present in the right most lanes. As you can see the different attributes of the table name for a lane are separated by '_'. This is used to differentiate the roads and lanes during rendering. The above example would generate a table name as 'I80_EASTBOUND_1' for a road with multiple lanes or 'I80_EASTBOUND_1_R' for a road with only 1 lane.

A step by step guide to importing data into ArcGIS to build a geodatabase is described in Appendix A.

2.4 Conclusion

Following the procedure described above, we have built a lane-level map, with decimeter accuracy, of the surveyed area and stored it in a geodatabase in ArcGIS. This database is then exported into an MS SQL Server database to allow real time data access.

Chapter Three: Re-Creation of Maps in Real Time for Navigation

3.1 Introduction

What are Real Time Systems?

Real Time Systems are systems in which the outcome is guaranteed in a fixed amount of time. This time could be 10 minutes or 10 milliseconds. As long as it is guaranteed to complete an assigned task before a deadline it is a real time system. This time varies depending on the application.

Why do we need real time navigation systems?

The commercially navigation systems available in the present day help us plan the route of our journey, and re-route us in case of human error resulting in missing a turn. Some systems also give us information on the number of lanes on the road we are travelling on and direct us to certain lanes for safe and comfortable driving. Some ADAS systems also give us information on the current lane the vehicle is driving along and have lane departure warning [26]. But what these systems lack is the ability to inform the driver about the vehicle's current position within the lane. We have

discussed in Chapter 2, that we use the Rover to acquire positional data with centimeter accuracy. Using this data along with the lane-level geodatabase with decimeter accuracy we can convey to the driver his position in the lane. As an additional feature we will also calculate the predicted vehicle path as discussed in section 3.9

To improve vehicle safety by informing the driver about his position within a lane, it is ijmperative to also deliver this information fast enough so that the driver has enough reaction time. Also such a system would need a high refresh rate as circumstances can change quickly. For example, a vehicle travelling at 65 mph covers a distance of 2.9 meters per second. If we design system which updates information to the driver at 10Hz, he would have travelled distance

$$d = \frac{290cm}{10} = 29cm$$

before he gets updated with his position. For our evaluation purpose we refreshed our system at this rate. Also, our designed system flagged the user if it could not deliver information at this rate. Hence, we designed a soft real time navigation system.

3.2 Our Approach

To deliver relevant information to the driver, our display screen would require the following:

1. An object representing the vehicle on screen.
2. A map of the region around the vehicle representing lanes.
3. A compass showing us the direction of motion of the vehicle.
4. A predicted path and a predicted vehicle position for the object.
5. Textual data representing the vehicle state and the integrity of our data.

To display the above listed items we had considered a few graphics libraries such as Qt, DirectX, Flash and OpenGL. After a brief evaluation of their features and complexity, and discussion with students who had used them, we chose OpenGL for its processing speed and comparatively easy learning curve. We chose to write our program on a Windows based platform in Visual Studio to make use of the MSDN libraries for SQL Server database access. I initially wrote the program in C#, but switched to C++; since I needed help debugging the program and it was the language Prof. Farrell knew.

To deliver the above listed information on screen we require two sets of data:

1. Vehicle State (Position, Velocity, Acceleration).
2. Lane-level map of the region near the vehicle.

We receive the vehicle state from a CPDGPS aided INS system, similar to the one used in data collection. This gives us the vehicle position with centimeter accuracy. This data is transmitted to our display program from the Rover via the serial port. We discuss the vehicle state transmitted and serial port connection in detail in section 3.3.

Once we know the current position of the vehicle, we query the database built in Chapter 2 and receive data representing the lane-level map of the surrounding region. We then process this data to display the required items to the driver. A flowchart of this process is shown in figure 4. The rest of this chapter discusses the various steps taken in order to construct our graphical display.

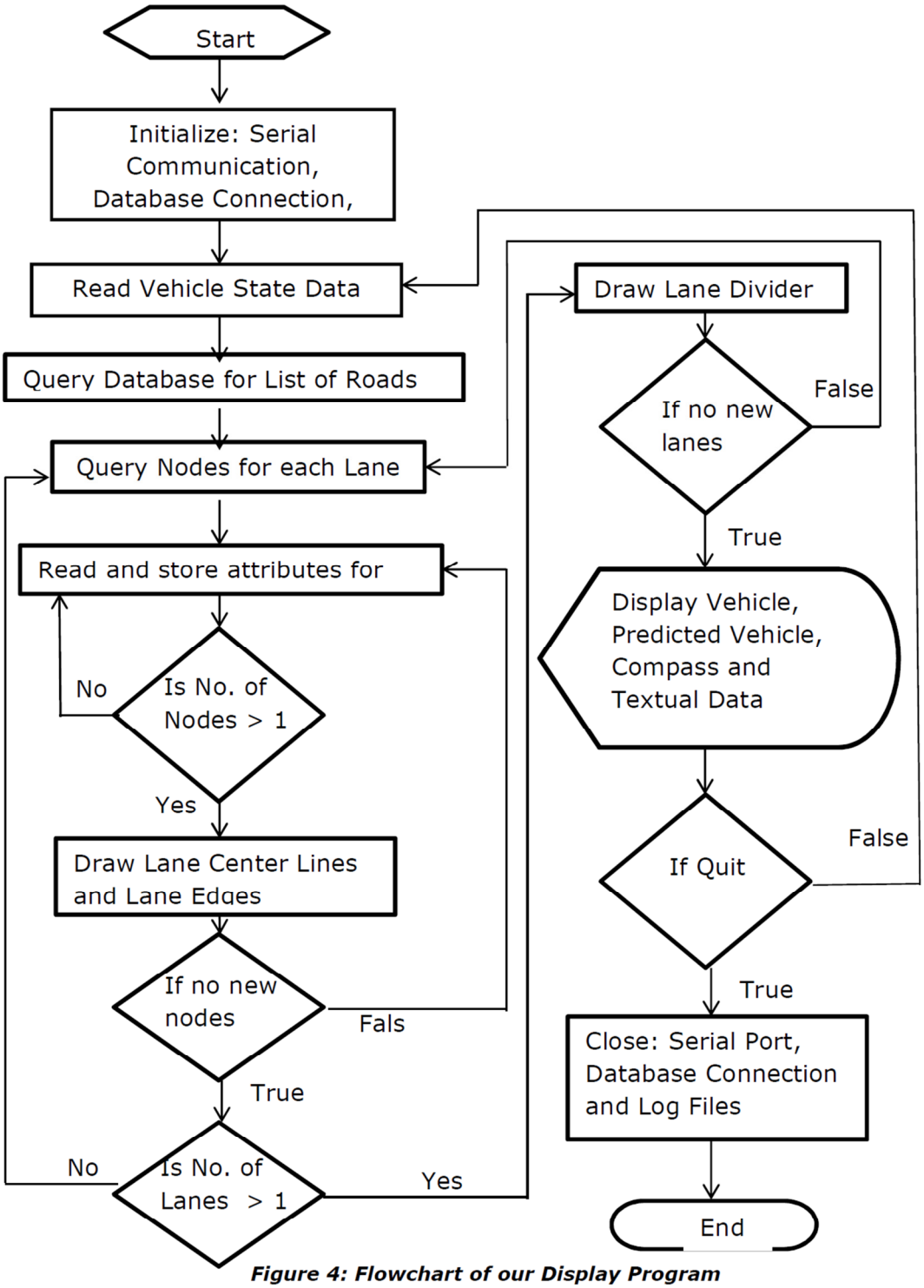


Figure 4: Flowchart of our Display Program

3.3 Current Vehicle State

The current vehicle state is transmitted to our display program from the rover. This data is then stored in an object of class 'Vehicle_Data'. These steps are discussed in the following subsections.

3.3.1 Data Transmission and Flow Control

We transmitted vehicle data from the rover to our computer through the serial port operating at a Baud rate of 38400 bits/sec, 8 data bits and 1 stop bit.

A vehicle data packet consisted of 39 bytes. The first two bytes were header bytes fixed as 'AA' and 'BB', followed by 36 bytes containing vehicle information. The last remaining byte was the checksum. The checksum was a simple sum of the 36 bytes preceding it.

The 36 vehicle information data was distributed in the following manner:

Bytes No.	**Total Bytes**	**Data Type**	**Vehicle Parameter**	**Units**	**Accuracy Level**
1-8	8	Double	Latitude	Radians	Cm
9-16	8	Double	Longitude	Radians	Cm
17-20	4	Float	Altitude	Meters	Cm
21-24	4	Float	Yaw	Radians	.001 rad
25-28	4	Float	Vehicle Speed - North Component	Meters/sec	Cm/s
29-32	4	Float	Vehicle Speed - East Component	Meters/sec	Cm/s
33-36	4	Float	Rate of change of Yaw	Radians/sec	.01 rad/s

Table 4: Vehicle State data transmitted from rover to HMI.

3.3.2 Data Reception and Storage

The data packet as described in 3.3.1 is transmitted from the rover. It is received by our program and stored in a buffer. The buffer is then traversed to search for the header. Once found the next 36 bytes are added to create a checksum. If this sum is same as the 39^{th} bit of the received data packet we know our data is accurate. We then proceed to parse the vehicle data and store it. It will be used subsequently to query the database and display information.

We have written a class called 'Vehicle_Data' which holds all data pertaining to the vehicle. The data is read from the serial port by the function 'Read_veh_data()' contained in the class 'DataPort' which handles the serial port communication.

3.4 Reference Frames

The display window consists of:

1. A Navigation Box, within which we will display the digital map, vehicle position and the predicted vehicle path.
2. A compass with an arrow pointing to North.
3. Textual Data

The current position of our vehicle in the display window is always fixed, while the lane-level map and the predicted path are redrawn continuously.

Figure 5 shows the reference frames related to our display screen. These frames are defined as follows:

Body Frame: Assuming that the roll and pitch of the vehicle are equal to zero, the body frame x-y plane is parallel to the geodetic tangent plane with the origin defined by the vehicle position.

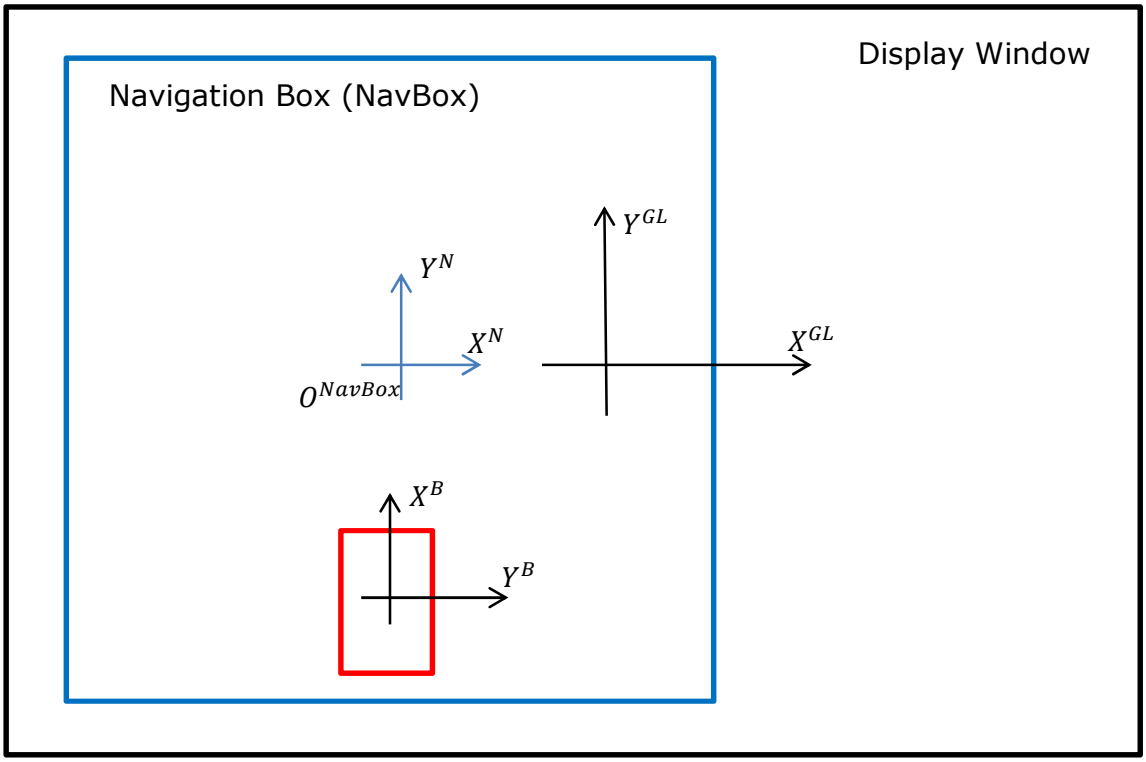


Figure 5: Reference frames in the display program

GL Frame: The GL frame is 2-D frame on the display window with its origin at the center of the display window. It has its x co-ordinate pointing to the

right of the screen. OpenGL uses certain units to transform desired co-ordinates to pixel units. We address these units as GL units for our program.

NavBox Frame: The NavBox frame has the same orientation as the GL frame and has its origin at the center of the navigation box. The box is drawn in blue. This NavBox is drawn such that it displays the map for a distance of 27 m. in the direction of motion of the vehicle and 20 m. in the direction perpendicular to it.

The NavBox is centered at a fixed location [-10, 0] GL units in the GL frame with its x and y axis aligned to that of the GL frame. Hence the transformation from the NavBox frame to the GL frame is given by

$$\begin{bmatrix} X^{GL} \\ Y^{GL} \end{bmatrix} = \begin{bmatrix} X^{N} \\ Y^{N} \end{bmatrix} + \begin{bmatrix} -10 \\ 0 \end{bmatrix}$$

This fixed location is stored in the 'NAVBOX_CENTER_REF' array in our program.

In the NavBox frame the current vehicle position is fixed at [0, -30] GL units and the x-axis of the body frame aligns with the y-axis of the NavBox frame. Therefore, points in the body frame are transformed to the NavBox frame as

$$\begin{bmatrix} X^{N} \\ Y^{N} \end{bmatrix} = \left(R_B^N * \begin{bmatrix} X^{B} \\ Y^{B} \end{bmatrix} \right) + \begin{bmatrix} 0 \\ -30 \end{bmatrix}$$

where,

$$R_B^N = \begin{bmatrix} \cos(\pi/2) & \sin(\pi/2) \\ \sin(\pi/2) & \cos(\pi/2) \end{bmatrix}$$

3.5 Database Query for Lane-level maps

The database queried in two steps.

1. Query database to get a list of nearby roads/lanes.
2. Query each lane, in order to retrieve nodes.

These steps are discussed in the next two sub-sections.

3.5.1 Creating Database Query For Nearby Roads/Lanes

The goal of the first database query is to get a list of roads/lanes which are in close proximity to the vehicle. The database contains a table called 'SDE_layers' which contains the maximum and minimum values of the latitude and longitude of each lane present in the entire database.

We create a query to fetch the roads in a box of size GEOBOX_LAT x GEOBOX_LON (apprx. 100m x 100m in our program) with the vehicle at its center. The following line of code creates this query string.

```
String ^query = "SELECT table_name FROM SDE_layers WHERE
maxy >= "+ Convert::ToString((LLA_veh[0]-GEOBOX_LAT/2)*rad2deg) + "AND
miny<="+Convert::ToString((LLA_veh[0]+GEOBOX_LAT/2)*rad2deg) + "AND maxx >=
"+Convert::ToString((LLA_veh[1]-GEOBOX_LON/2)*rad2deg) + "AND minx <=
"+Convert::ToString((LLA_veh[1]+GEOBOX_LON/ 2)*rad2deg);
```

'GEOBOX_LAT' and 'GEOBOX_LON' are constants defined in the class 'SQL_Conn'. The above query is created in the initial part of the

GetDrawData() function, which handles the entire process from querying all data and then calling functions to display it on screen.

3.5.2 Querying Database For Individual Lane Data

Now that we have a list of lanes in close proximity to the vehicle, we need to query each lane and draw it to the display window. We store the lane names in an array road_names[] and query all data from the tables for the related lanes using a query

"SELECT * FROM road_names[i]".

3.6 Interpretation and Filtering of lane level data

A table containing information for a particular lane is queried and the information for each node is read sequentially. Once we have two nodes available we use the Hermite polynomial to calculate points in between these two nodes using the function 'ComputeHermiteCurve()'and render a dashed line using the function 'DrawArc()' which invokes the required functions from the OpenGL library. During this process we also store the tangents at each of the generated points. This line depicts the center of the lane and the color of the dashed line depicts the vehicles direction of motion in comparison to lawful direction.

If the yaw of the vehicle is within +/-1 radian of the trajectory, we assume the lane to be in the same direction of motion as the vehicle and draw it green; otherwise we draw it as red. Therefore lanes for traffic travelling in the opposite direction are always red.

We continue drawing each section of the lane until the entire lane is drawn and move on to the next lane.

3.7 Drawing Lane Edges and Lane Dividers

The lane name is parsed to determine the road name and the lane number. If the current lane is lane number 1, we draw the left edge of the lane by projecting each of the calculated points between two nodes to the left at an angle perpendicular to the trajectory tangent at that point and by a distance which is half of the lane width. This is mathematically expressed as

$$Pt_{left\_edge} = Pt + \left(\frac{lw}{2}\right) * \begin{bmatrix} -\sin(psi) \\ \cos(psi) \end{bmatrix}$$

Where Pt are the points calculated on the lane trajectory and 'psi' is the yaw of the trajectory at that point.

If it is the right edge we project and draw on the right hand side as well.

$$Pt_{right\_edge} = Pt + \left(\frac{lw}{2}\right) * \begin{bmatrix} \sin(psi) \\ -\cos(psi) \end{bmatrix}$$

The lane edges are drawn as solid yellow lines.

If we have two lanes on the same road, we want to draw the lane divider line between them. To achieve this we first need to calculate the lane width. The lane width is computed by finding the distance between the points on each trajectory that is closest to the vehicle. This computation is described in section 3.8.

When we have the reference points at minimum distance from the vehicle for two adjacent lanes, we calculate the distance between these points and determine this to be the lane width.

We then draw the lane divider lines as dashed white lines connecting the points projected from the rightmost lane. These points are computed as

$$Pt_{lane\_divider} = Pt + \left(\frac{lw}{2}\right) * \begin{bmatrix} -\sin(psi) \\ \cos(psi) \end{bmatrix}$$

3.8 Compute Point on Trajectory at Minimum Distance to the Vehicle

Figure 6 shows a trajectory γ(s) defined by nodes A and B with A = γ(0), where 's' is the arc length travelled along the trajectory from A towards B.

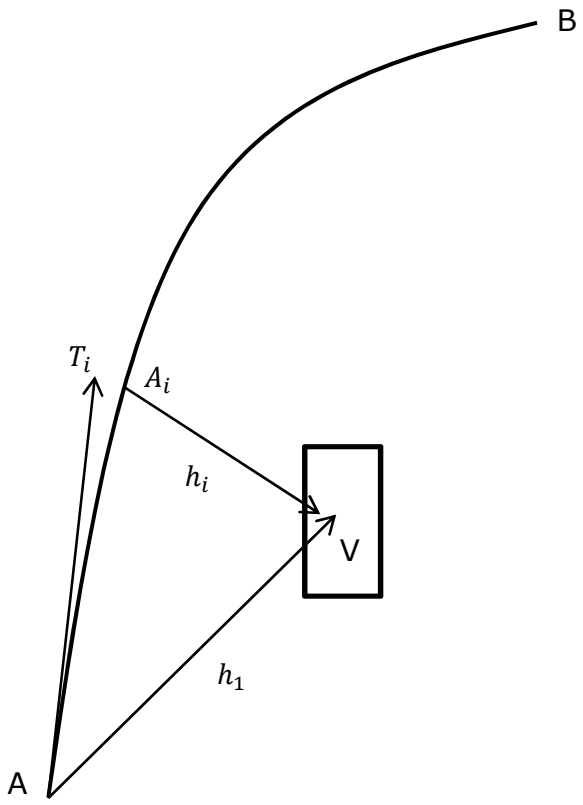


Figure 6: Computing a point on a trajectory which is at min. distance to the vehicle.

We start by defining,

$$A_1 = A, with\ s_1 = 0$$

Let, $$T_i = \frac{\delta\gamma(s)}{\delta s}|_{s=s_i}$$

$$h_i = V - A_i$$

Where *V* is the vehicle position and A_i is a point on the trajectory.

$$s_{i+1} = s_i + T_i * h_i$$

$$A_i = \gamma(s + i)$$

We repeat the above steps till $(T_i * h_i)$ < 0.01 m.

Hence, the last point $A_i$computed is the closest point on the trajectory to the vehicle. This algorithm is implemented in 'GetArcLengthatMinDist()' in the Traj_Data class.

3.9 Display current and predicted vehicle position

The current vehicle position is always fixed at a point on the screen. The lanes are drawn in reference to it. The vehicle is always travelling towards the top of the screen. The dimensions of the vehicle are specified by constants 'VEHICLE_LENGTH' and 'VEHICLE_WIDTH'. The current vehicle position is depicted by a red rectangle.

The predicted vehicle position is drawn in blue and the path followed by the vehicle to reach there is drawn as a white line. The vehicle is shown 3

seconds or 20 meters, whichever is less, in the future. The path followed by the vehicle is calculated as follows:

We first calculate the distance travelled by the vehicle in 3 seconds as

Distance (Dist) = Speed of vehicle (V) x 3 secs

We compare this distance with 20 meters and determine the time steps at which we want to draw our path. If Distance > 20m. then we calculate the time step as

$$dt = \frac{20}{V * N}$$

where, N is the number of points in between the current vehicle position and the predicted one. If not we determine the time steps as

$$dt = \frac{3}{N}$$

Define, $yw_0 = eps$

Where, 'eps' is the angle between the vehicles direction of motion and the direction at the point on the trajectory which is nearest to the vehicle

Now we calculate the predicted path using the algorithm:

For i=1:N

$$yw_i = yw_{i-1} + g_z * dt$$

$$s_i = s_{i-1} + dt * V * \cos(yw_i)$$

$$d\_s_i = d\_s_{i-1} + dt * V * \sin(yw_i)$$

Where, g_z is the rate of change of the yaw of the vehicle; yw_i represents the instantaneous yaw of the vehicle and $[s_i, d\_s_i]$ is the predicted position of the vehicle relative to the Frenet frame, which is a tangent plane that has its

x-axis aligned to the tangent at a point on the trajectory nearest to the vehicle and its origin at the position of the vehicle. In Figure 7, [s, d_s] represent the Frenet frame with 's' parallel to the Tangent 'T' at a point on the lane centerline closest to the vehicle.

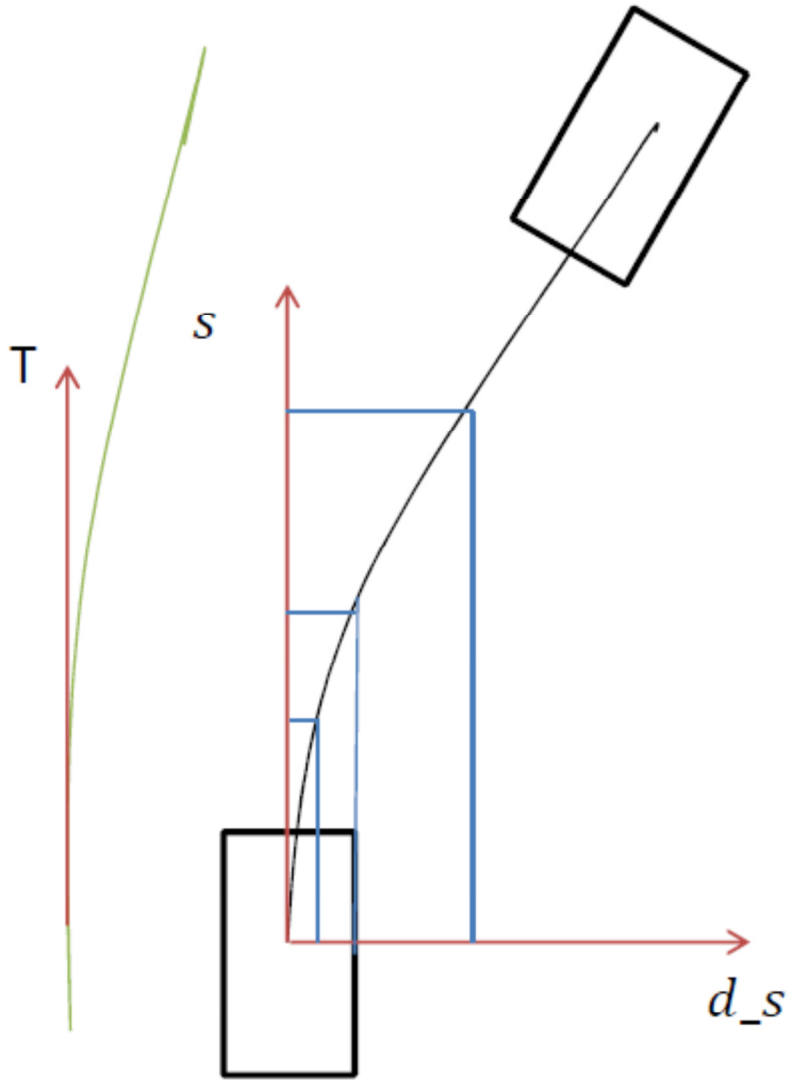


Figure 7: Compute Predicted Vehicle Path

This algorithm is implemented in the function 'DrawPredictedPath()'.

3.10 Display Compass

The compass is drawn in the right hand corner of the screen as a blue circle with a red arrow which is always pointing to the north. This is implemented in 'DrawCompass()'.

3.11 Display Textual Data

Textual data is rendered on the right hand of the screen and displays the vehicle state. It also shows us the status of our SQL Database Connection and the integrity of our data transmitted from the serial port.

3.12 Conclusion

The system designed by us displayed to the driver the vehicles current and predicted vehicle position relative to the lanes on the road the vehicle is currently on. We updated our vehicle state and refreshed our display at 10Hz.

Chapter Four: Applications

4.1 Snow Plow Guidance

The navigation system described above was designed as a part of a Snow Plow Guidance Project for California Transportation Systems. In June 2010, we gave a final demonstration to CALTRANS of our system. It was a two week process wherein we setup our base station and surveyed all lanes along the I80 between Donners Lake exit at the east end and the Big Bend/Rainbow Road exit at the west end in Week 1. We processed this data and built a geodatabase at the end of the week. Week 2 was spent editing the database, replacing sections which were not believed to be of decimeter accuracy. The results for the same projected in Google earth are shown below. Figure 8 shows a map in of the region surveyed on I-80.

Figure 9 shows us two trajectories in each direction for a smaller area.

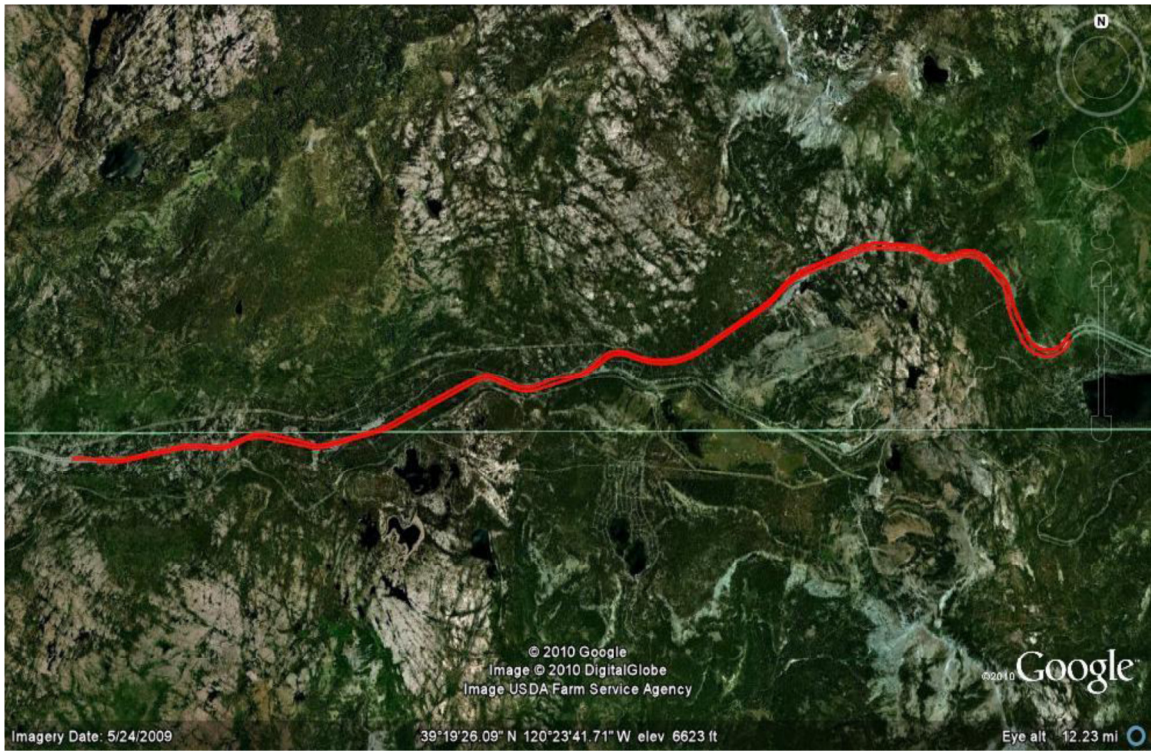


Figure 8: Region on I-80 between Donners Lake Exit and Big Bend Exit.

Figure 9: Lane-level trajectory curve fit on I-80

In figure 10, we can see the display window on the computer screen showing us the current vehicle position as a red rectangle. It shows us that we are near the right edge of the lane which can be confirmed by looking ahead from the windshield. The predicted vehicle position can be observed as a blue rectangle and the path followed as a solid white line.

Figure 10: A photograph of the navigation system working in real time

Chapter Five: Conclusions and Future Work

We were successful in developing an Advanced Driver Assistance System which informed the driver of his position within the lane and the predicted vehicle path. In order to do so we built lane-level digital maps with decimeter accuracy and accessed them in real time.

The system we built operated in real time but we could not guarantee this operation. A problem we could see arising is if our database access was slow due to large amounts of data. To solve this issue, we could write a multi-threaded program instead of our sequential program with one thread dedicated to receiving data from rover, another to querying data from the geodatabase and a third to display this data. In effect if we have a large query window, i.e. we query an area of a 100m x 100m around the vehicle and we display only 30m x 30m, we could query the database at lower rate and still maintain a real time system.

Improvements in Geodatabase

Design: The geodatabase design could be improved to incorporate much larger datasets. In our design we have a separate table for a list of nodes for

each lane. We could add another layer to first query roads and then lanes within each road.

Adding node attributes: We could store more attributes in the geodatabase like Landmark storage for better position estimation when used with vision systems or laser scanners. If we have high performance computing systems we could also make use of storage of images along the road. This would be especially useful in Snow Plow Guidance during whiteouts.

Speed Assistance

We could assist users by suggesting safe speeds at that particular time on the roads that one is on. This would be helpful in hilly areas where we need to continuously monitor our speeds through turns. It could also suggest acceleration or deceleration to drivers to improve fuel economy.

3D Rendering

The graphics displayed by us our two dimensional. We could use 3D graphics to improve user experience which would further lead to rendering of the neighboring terrain. We could also overlay satellite imagery onto our map to provide more information to the driver.

References

[1] Jay A. Farrell, Aided Navigation: GPS with High Rate Sensors, McGraw Hill, 2008

[2] J. Allison Butler, Designing Geodatabases for Transportation, ESRI Press, 2008.

[3] A. Chen, A. Ramanandan, J. A. Farrell, "High-precision lane-level road map building for vehicle navigation: nodal approach," *IEEE/ION PLANS 2010*, May, 2010**.**

[4] J.A. Farrell, "Highway Lane Curve Fitting: Nodal Approach", July 2008.

[5] Fuquan Pan, Lixia Zhang, Fengyuan Wang, "GIS and GPS Based Vehicle Guidance System", ICICTA, 2008.

[6] J.A. Farrell, "Computation of Vehicle Control State: Nodal Approach", July 2008.

[7] Spatial Data Standards and GIS Interoperability, ESRI, White Paper.

[8] R.H. Bartels, J.C. Beatty, and B.A. Barsky, "An Introduction to Splines for Use in Computer Graphics and Geometric Modeling. Morgan Kaufman, 1987.

[9] J. M. Clanto, D. M. Bevly, and A. S. Hodel, "A low-cost solution for an integrated multisensor lane departure warning system," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 1, pp. 47–59, March 2009.

[10] J. A. Farell, H.-S. Tan, and Y. Yang, "Carrier phase GPS-aided INS based vehicle lateral control," *Journal of Dynamic Systems, measurement and Control*, vol. 125, pp. 1–15, Sept 2003.

[11] J. A. Farrell and M. Barth, *The Global Positioning System And Inertial Navigation*. McGraw-Hill, 1998.

[12] M. F. Goodchild, "GIS and transportation: Status and challenges," *GeoInformatica*, vol. 9, no. 1, pp. 59–87, July 2000.

[13] S. Mortensen, "USDOT's demonstration and deployment programs on vehicle assist and automation for bus rapid transit," in *Intelligent Transportation Systems, 2009. Proceedings*, St. Louis, MO, Oct 2009.

[14] J. Saches-Reyes and J. M. Chac´on, "Polynomial approximation to clothoids via s-power series," *Computer-Aided Design*, vol. 35, no. 14, pp. 1305–1313, March 2003.

[15] I. Skog and P. H¨adel, "In-car positioning and navigation technologies - a survey," *IEEE Transaction on Intelligent Transportation Systems*, vol. 10, no. 1, pp. 4–21, March 2009.

[16] J. WANG, S. Stefan, M. Klaus, O. Roland, J. Armin, and P. Thomas, "Lane keeping based on location technology," *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, no. 3, pp. 351–356, September 2005.

[17] C. K. H. Wilson, S. Rogers, and S. Weisenburger, "The potential of precision maps in intelligent vehicles," in *IEEE International Conference on Intelligent Vehicles*, October 1998, pp. 419–422.

[18] G. Zhang and C. Wilson, "An integrated DGPS/DR/map system for vehicle safety applications," in *Proceedings of the 2000 National Technical Meeting of the Institute of Navigation*, January 2000, pp. 253–257.

[19] S. Schroedl, K. Wagstaff, S. rogers, P. Langley, and C. Wilson, "Mining GPS traces for map refinement," *Data Mining and Knowledge Discovery*, vol. 4, no. 2, pp. 127–139, June 2000.

[20] J. A. Farrell, "Land Vehicle Path Following Control", July 2009.

[21] J. Rowell, "Applying map databases to advanced navigation and driver assistance systems," *Journal of Navigation*, vol. 54, no. 3, pp. 355–363, Sep. 2001.

[22] Felipe Jiménez *, Francisco Aparicio, Gonzalo Estrada "Measurement uncertainty determination and curve-fitting algorithms for development of accurate digital maps for advanced driver assistance systems", Instituto Universitario de Investigación del Automóvil (INSIA), Universidad Politécnica de Madrid, Campus Sur UPM, Carretera de Valencia km 7, 28031 Madrid, Spain

[23] Bendafi, H., Hummelsheim, K., Sabel, H., van de Ven, S., "Classification of data capturing/production techniques. NextMap Project Deliverable D 3.1", 2000.

[24] Miles, J.C., Chen, K.," ITS Handbook", second ed. PIARC, 2004.

[25] Yerpez, J., Ferrandez, F., "Road characteristics and safety, Identification of the Part Played by Road Factors in Accident Generation". INRETS, Arcueil, 1986.

[26] EDMap Consortium, "Enhanced digital mapping project Final Report", 2004.

[27] eSafety Forum, "Digital Maps Working Group Final Report". European Commission (eSafety Forum), Brussels. 2005.

[28] Kang Li, Han-Shue Tan , James A. Misener , J. Karl Hedrick, "Digital Map as a Virtual Sensor – Dynamic Road Curve Reconstruction for a Curve Speed Assistant", Department of Mechanical Engineering, University of California, Berkeley, USA.

[29] Akshay Morye, "Intelligent Roadways and GIS", 2008.

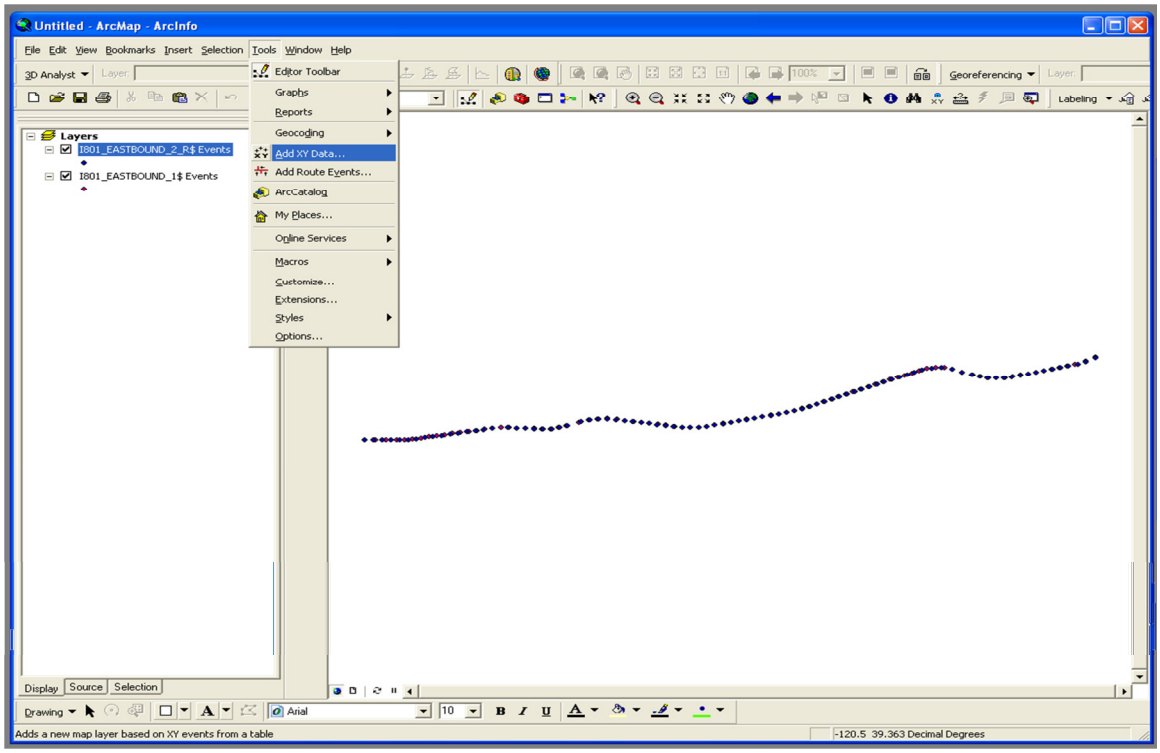
Appendix A

Steps to build a geodatabase using ArcGIS from a list of XLS files, each containing Nodes and Its Attributes for a lane.

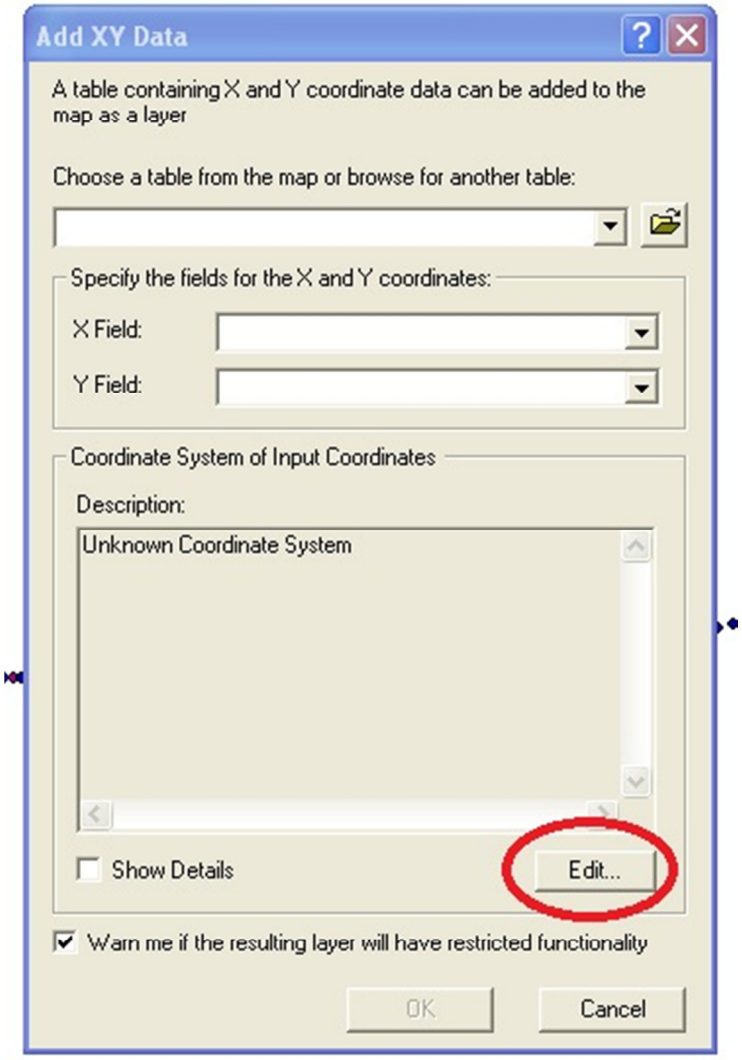
1. You will need ArcGIS installed on your computer running with the ArcINFO license suite.

2. Browse to the ArcGIS Menu in the Start Menu panel and open ArcMap. Once the program is open you will be asked if you want to open an existing map or create a new one. We will create a new map.

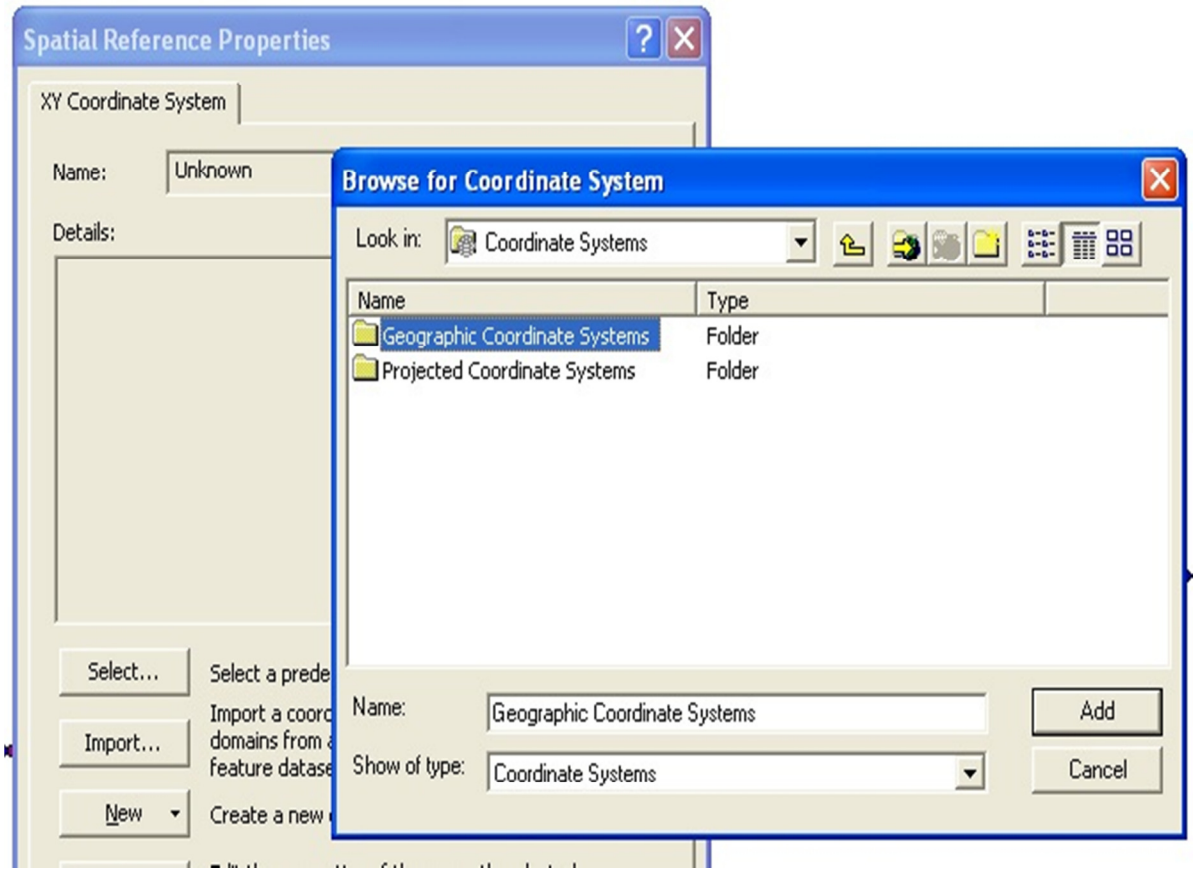
3. Next go into the Tools Menu and select the 'Add XY Data' tool as shown below



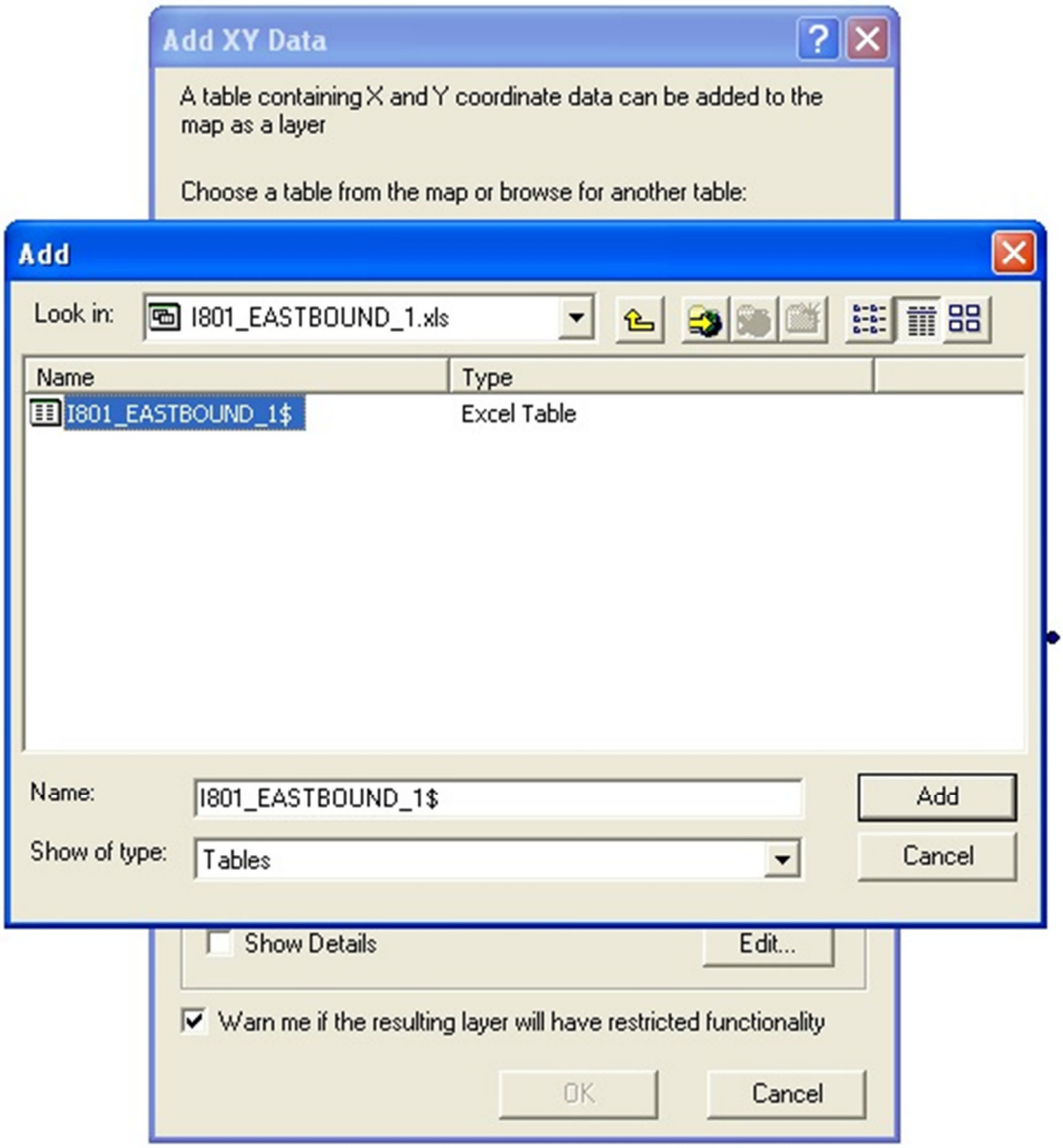
4. Now we will have a window as shown below.



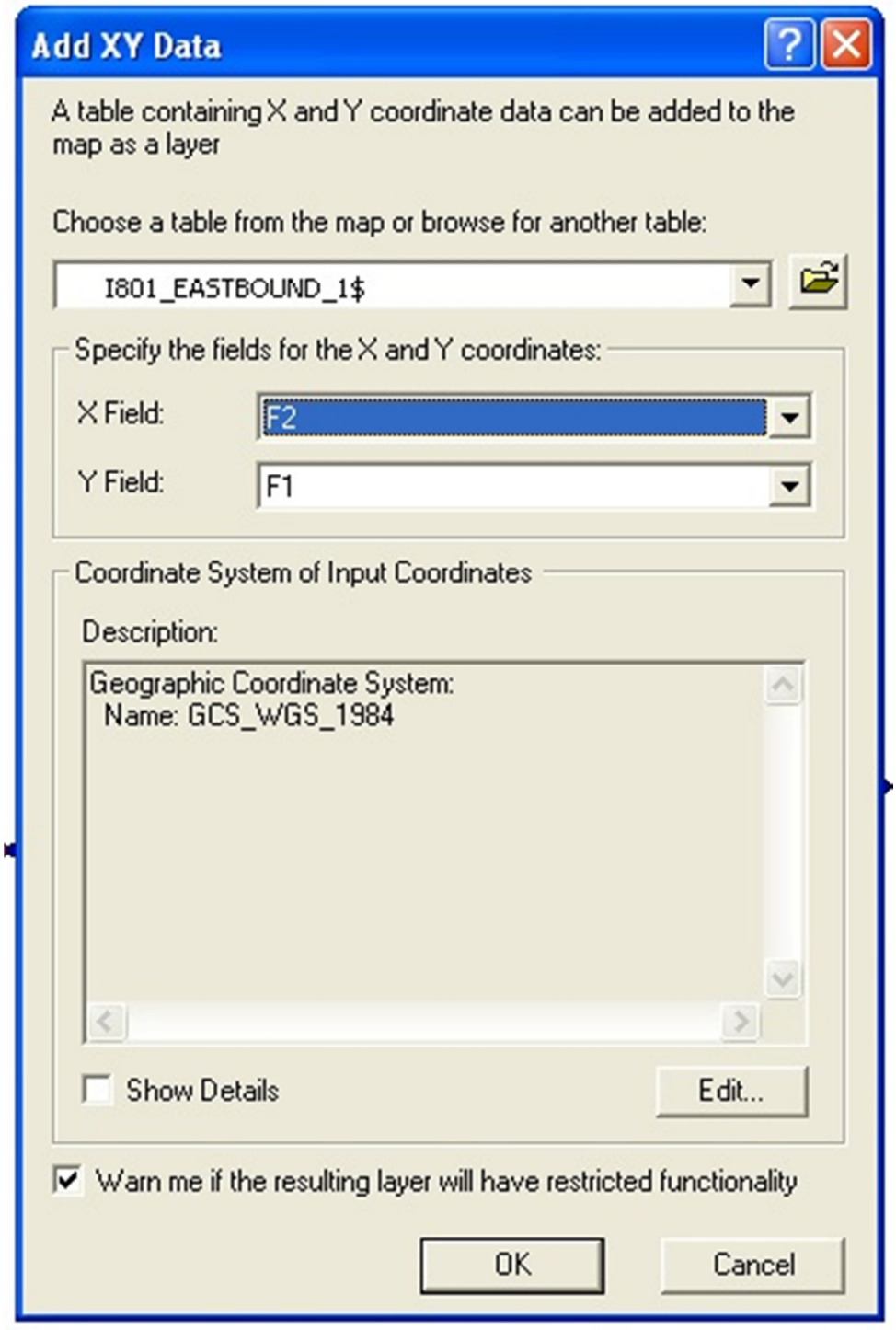
5. Click edit and select the WGS 1984 parameters as shown below


Spatial Reference Properties
XY Coordinate System
Name:
Unknown
Details:
Select...
Import...
New
Browse for Coordinate System
Look in:
Coordinate Systems
Name
Type
Geographic Coordinate Systems
Folder
Projected Coordinate Systems
Folder
Name:
Geographic Coordinate Systems
Show of type:
Coordinate Systems
Add
Cancel

6. Next we browse to the XLS file we want to add and select the table within it as shown below



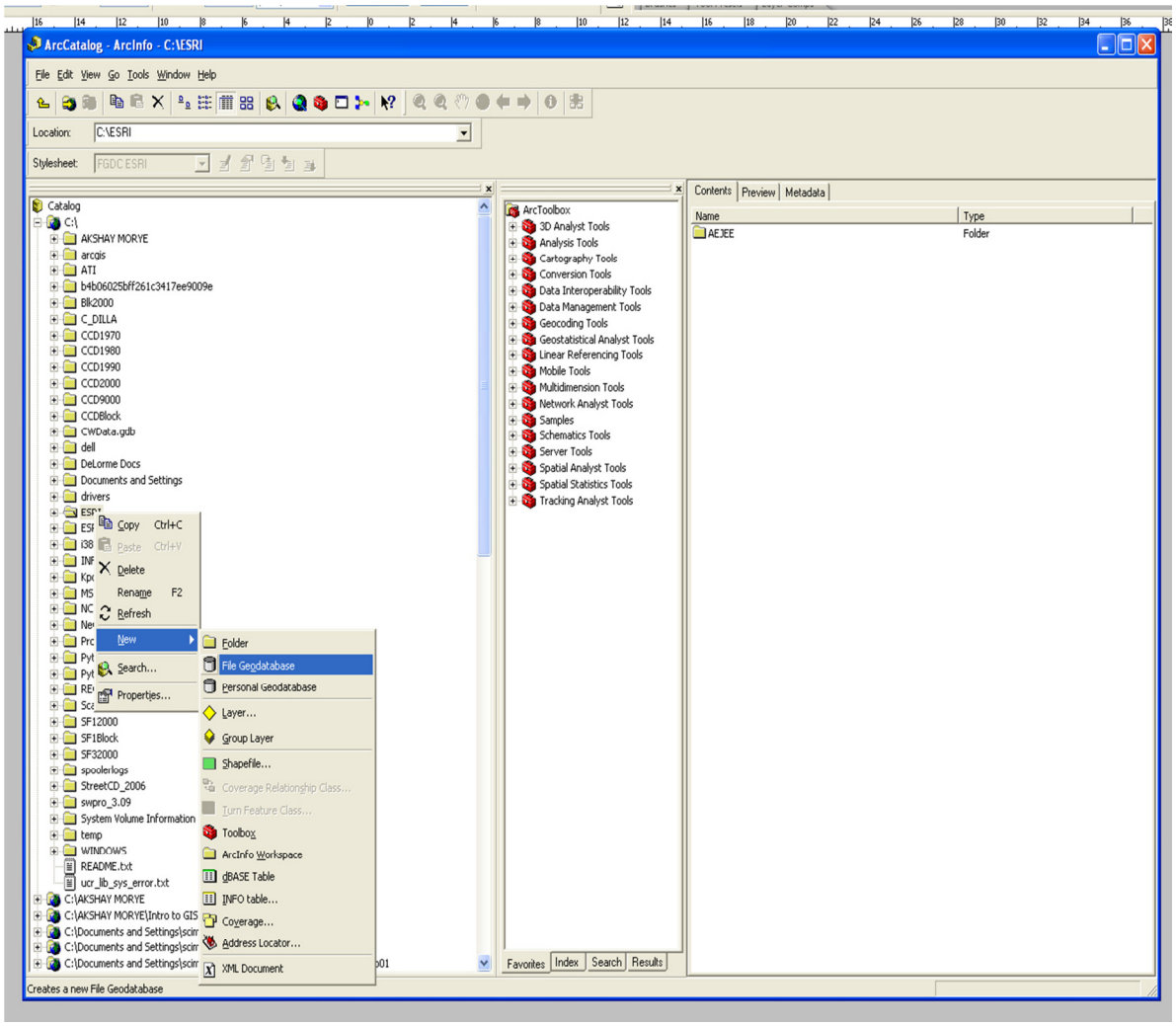
7. We select the X data (Longitude) as F2 (Field 2) and Y data (Latitude) as F1 (Field 1). The prompt window thus looks like follows:



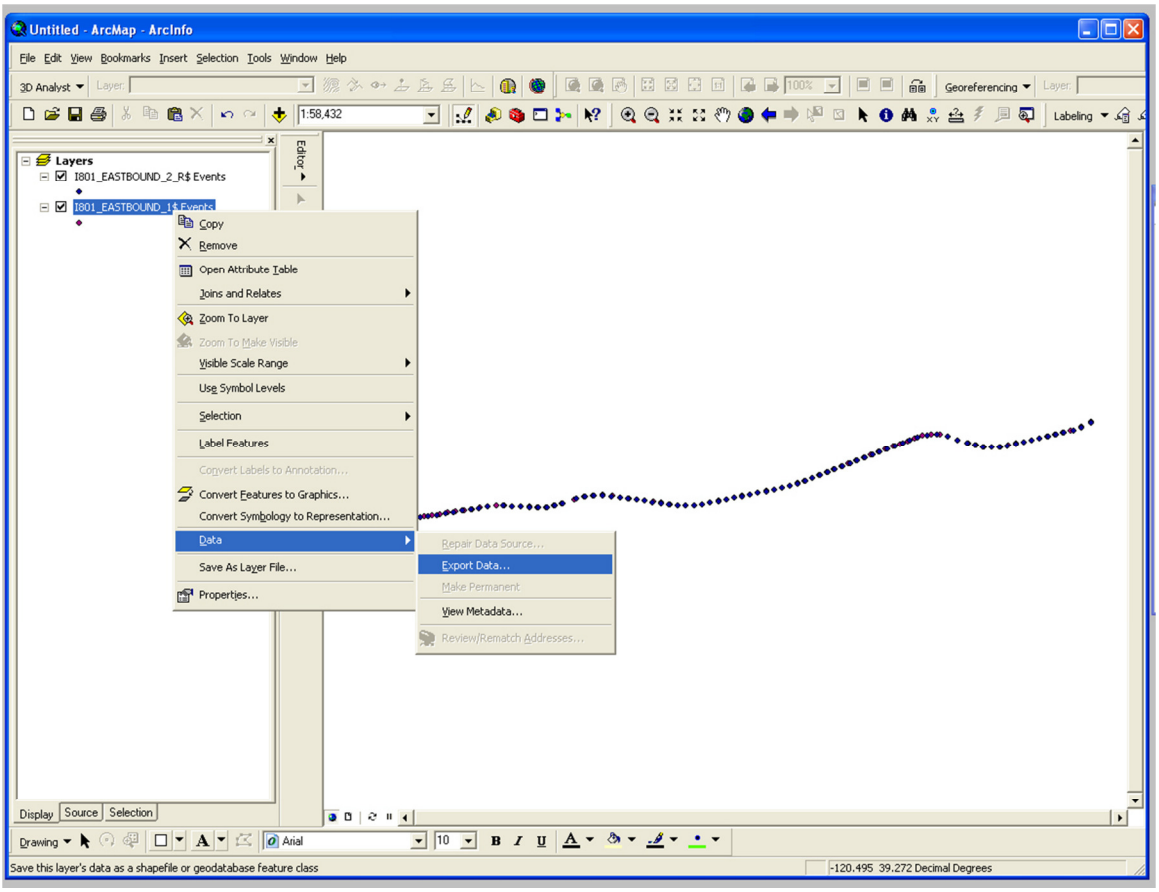
8. Next we start ArcCatalog from the Start Menu.

9. We browse to the folder we want to setup the new geodatabase in.

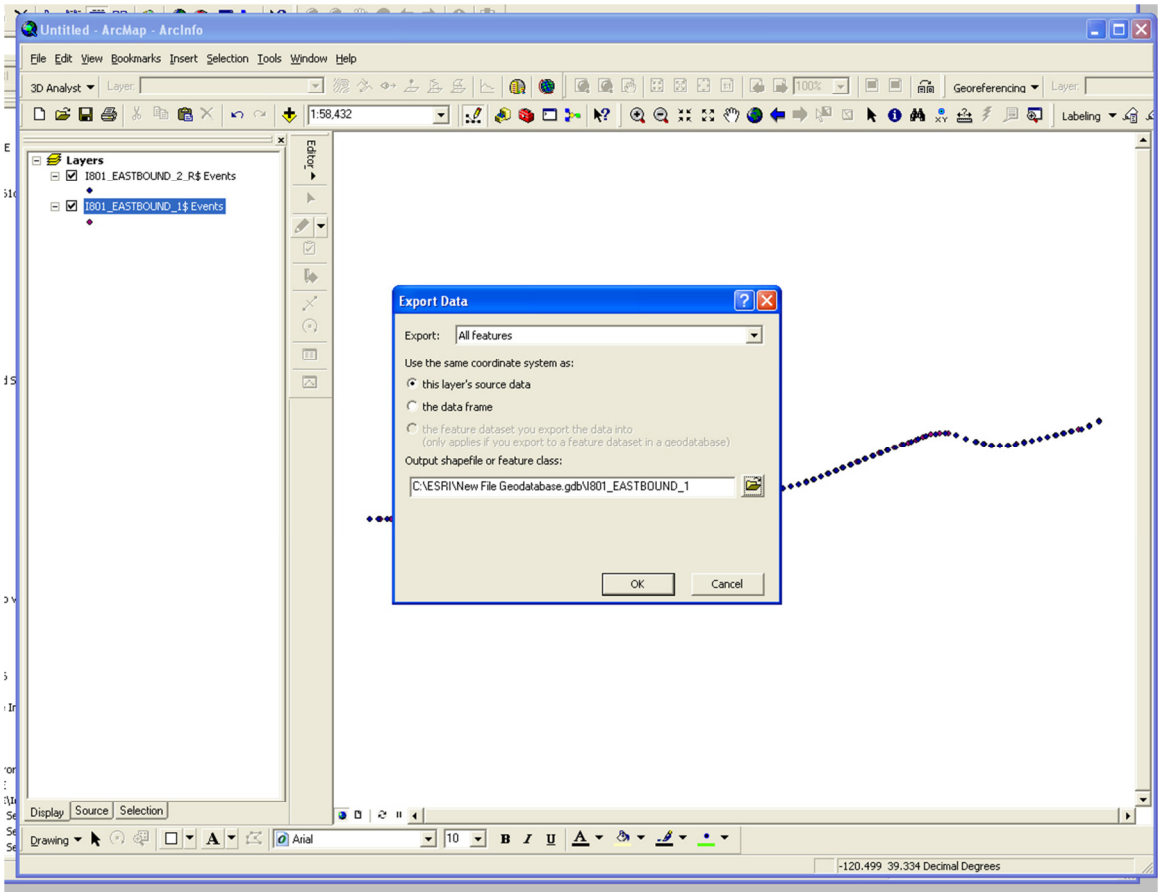
Then we right-click and select New --→ File Geodatabase as shown below.



10. Now, we go back to ArcMap and right-click on the Traj$ files one at a time, and select Data ---→ Export as shown below



11. We browse the our geodatabase created using ArcCatalog and select it. When prompted if we want to add the trajectory in the geodatabase onto our map as a layer, we select yes.

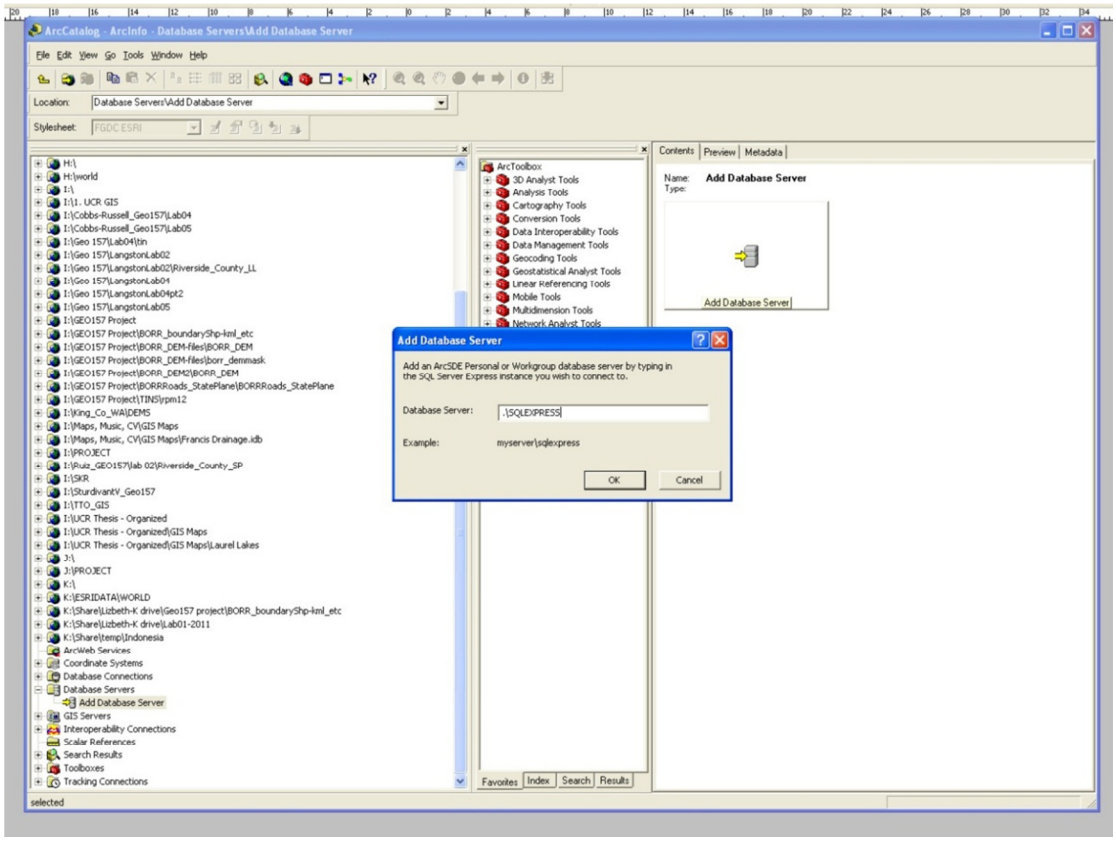


12. After this step we will have duplicate layers for each trajectory in our map, one from the geodatabase and one from the XLS table. We go ahead and remove the one from the XLS table.

Appendix B

Instructions to Transfer the Geodatabase in ArcGIS to Microsoft SQL Server.

1. We go to ArcCatalog and select the Add Database Server as shown below.



We add our SQL Server database name for example ".\SQLEXPRESS" to ArcCatalog if this is the first time we are transferring a geodatabase.

2. Next we right click on the server and create a new Geodatabase.

3. Lastly, we right click on this new geodatabase on the SQL Server and select to Import. We then browse to our geodatabase created from maps and select to import all feature classes.

4. This completes the transfer from the ArcGIS database to the MS SQL Server.