10/06/00 jc930 U.S. PTO

10-10-00 A/prov

PROVISIONAL APPLICATION COVER SHEET [37 CFR 1.53(c)]

This is a request for filing a PROVISIONAL APPLICATION under 35 U.S.C. §111(b) and 37 CFR 1.51(a)(2)

Date : October 6, 2000
Docket No. : 40668/JEC/D465
EXPRESS MAIL NO. EL386999236US

jc714 U.S. PTO 60/238490 10/06/00

Mail to: BOX PROVISIONAL PATENT APPLICATION

INVENTOR(S)/APPLICANT(S) (LAST NAME, FIRST NAME, MIDDLE INITIAL, RESIDENCE (CITY AND EITHER STATE OR FOREIGN COUNTRY)

Enrico DiBernardo; Pasadena, California

Luis Goncalves; Pasadena, California

____ Additional inventors are being named on separately numbered sheets attached hereto.

TITLE OF THE INVENTION (280 characters max)

SYSTEM FOR CREATING AND USING A VISUAL RECORD OF A GEOGRAPHIC LOCATION COUPLED WITH POSITION INFORMATION

ENCLOSED APPLICATION PARTS

__95__ Specification (number of pages)
__21__ Drawings (number of sheets)
____ Small Entity Statement
____ Assignment
__X__ Other (specify): Return postcard

FEE AND METHOD OF PAYMENT

__X__ A check for the filing fee of $ 150.00 is enclosed.
The Commissioner is hereby authorized to charge any fees under 37 CFR 1.16 and 1.17 which may be required by this filing to Deposit Account No. 03-1728. Please show our docket number with any charge or credit to our Deposit Account. **A copy of this letter is enclosed.**

____ No filing fee enclosed.

The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.

__X__ No ___ Yes, the name of the U.S. Government agency and the Government contract number are:

Please address all correspondence to **CHRISTIE, PARKER & HALE, LLP, P.O. Box 7068, Pasadena, CA 91109-7068, U.S.A.**

Respectfully submitted,
CHRISTIE, PARKER & HALE, LLP

By _____
Josephine E. Chang
Reg. No. 46,083
626/795-9900

PROVISIONAL APPLICATION FILING ONLY

JEC/daa
DAA PASZ78137 1-*-10/6/00 10:52 AM

1. Database creation

Acquisition of video and geo-location information (Flow Chart 2)

Figure 2 shows a pictorial representation of the scanning vehicle. The scanning team consists of two people, a driver and a navigator[1].

The shell containing up to four cameras is made in lightweight and shock resistant material. It can be air shipped and quickly mounted on top of an SUV. The cameras inside the shell can be accessed through the vehicle sunroof.

In 202 the visual information is acquired in the form of digital video on magnetic tape support using 1,2 or 4 digital camcorders mounted on top of a vehicle[2]. The first two cameras film the side views and the other two cameras film the front and back view. These last two cameras are equipped with fish-eye lens for a wide-angle view of the street. In order to practice the invention at least one video camera looking sideways is necessary. In this case the vehicle would have to be driven twice through the same street in opposite directions. In order to acquire both side of the street at the same time, the second side camera can be employed. The two front and back looking cameras are optional.

In 201, the positional information is acquired by an inertial navigation system equipped with accelerometers, gyroscopes and a GPS receiver[3] (see Note1)

In 203 a driving path is defined in order to minimize the driving time necessary to acquire the video information for a certain area. Different driving strategies will be used depending on the number of video camera employed. The 1-camera acquisition scheme consists in driving around each block making always right turns (see Figure 1). Although many other schemes are possible, this scheme has the following advantages:

- Frequent turns allow to be more robust to errors in the position detection.
- In presence of deviations from a regular grid structure in the street layout, it decreases the likelihood of failing to drive through and acquire some street segments.
- Minimizes the amount of stop time at the intersections, in the states were right turns are allowed on a red light.

[1] In order to practice the invention, any mobile platform can be employed.

[2] The video information can be acquired also using analog cameras, recording on any type of storage means (optical, magnetic or silicon). In future embodiments, it may be desirable to include even more cameras. One could imagine, for example, using a duodecahedron of cameras to record all viewing directions.

[3] In order to practice the invention, positional information can be acquired using means other than an inertial navigation system or a GPS, such as computer vision based algorithms that allow to compute at any time the position of the vehicle with respect to an initial position, using the video information from one or more video cameras.

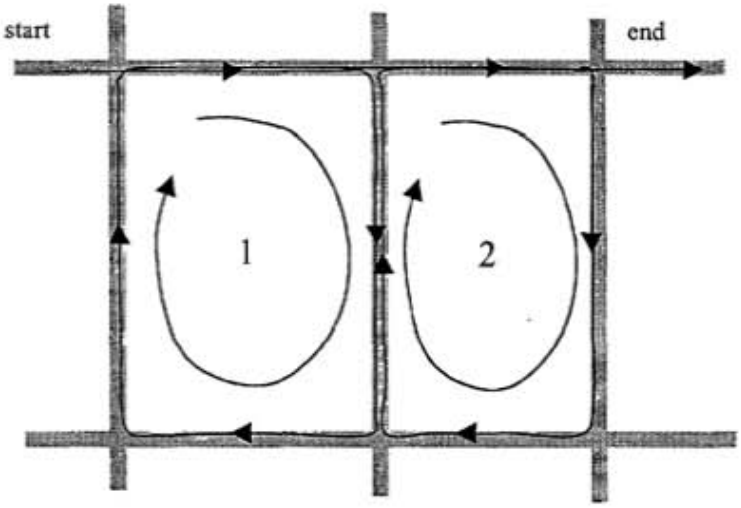


Figure 1: 1-Camera traversing scheme

In order to guarantee a sufficient resolution in the resulting images, the acquisition is done at an average speed of about 20 miles/hour. [4]

A laptop operated by the navigator computes the current position of the vehicle based on the information received from the GPS receiver and the accelerometers and provides vocal directions to the driver according to a predefined driving path. It also monitors and remotely controls the 4 camcorders through a LANC interface.

Figure 3 shows a block diagram of devices and their interconnections.

In 204 the positional information is recorded in a computer memory for further processing.

In 205 the video sequence is transferred to a media that allows random access to any particular frame in the sequence and to any video pixel (picture element) in such frame. In the present embodiment the video sequence is acquired in digital format on magnetic support and later transferred to a computer hard disk using a Fireware acquisition card. The video sequence is stored in the computer file system in the Audio-Video Interleaved format (AVI). The video stream in such file can be either the original DV format or can be compressed to MPEG for a more compact storage.

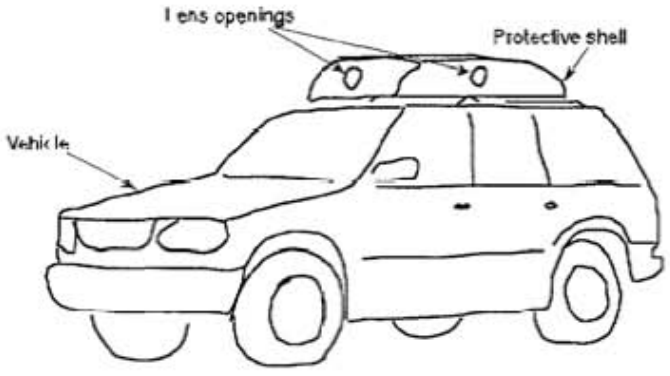


Figure 2: Scanning vehicle

[4] This speed limitation is imposed by the sampling rate of the video acquisition. Higher sampling rates would allow for faster acquisition speeds.

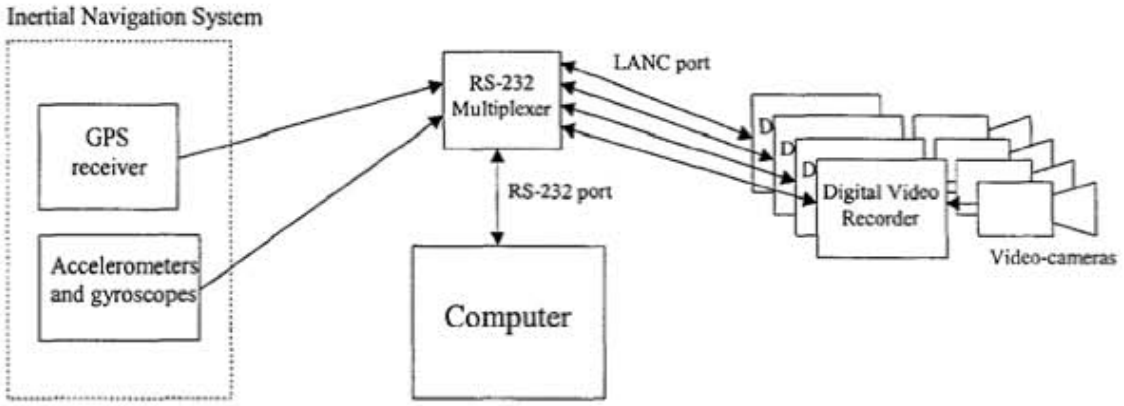


Figure 3: Acquisition System Diagram

Note 1. GPS Sampling rate is finite: The GPS receiver currently used provides position data at an approximate rate of 1 sample every 2 seconds. During acceleration and deceleration (especially during car starting and stopping), this rate does not provide enough samples to accurately define the position as a function of time curve (as defined in **Figure 4**). Currently, the path is estimated by linearly interpolating between the samples. This first-order approximation creates some distortions in the panorama. If the speed is overestimated, the panorama will be stretched (since we believe we are moving when in fact the video does not change correspondingly). If the speed is underestimated, the panorama will be squished (since we estimate that we do not move very much whereas the video has a larger motion).
A more accurate solution to this problem is to enhance the GPS position data with Inertial Navigation data. Accelerometers and gyroscopes can monitor the linear and rotational acceleration rates at 200Hz. This signals can be integerated to obtain position as a function of time, and can be combined with GPS signals to produce accurate position data with 200 samples per second. Inertial methods are accurate over short time frames due to the high sampling rate, but can accumulate integration error. GPS, on the other hand, is inaccurate at short intervals (due to the low sampling rate) but can be used to correct Inertial integration errors.

Segmentation and labeling of geo-location data (Flow chart 7)

Geo-location data consist of sequences of geographic coordinates (latitude, longitude) each labeled with the UTC time at which such coordinates where acquired.
The following process is applied to each one of the sequences.

In 701 the (lat, long) coordinates are converted into Mercator coordinates (cylindrical projection of latitude and longitude). A spline interpolation (cubic polynomial functions with local support) is obtained from the two-dimensional Mercator coordinates. The resulting spline function is parameterized in arc-length. A new sampling of coordinate samples is obtained from the spline function, sampling uniformly in arc-length with an arc-length increment of 1 meter or so. In order to detect the points in the new sequence where a turn was made, an operator is applied to the sequence that computes for each sample a value which is maximum in the center of a turn. Such operator computes the smaller of the two singular values of a 2xN matrix containing the N samples around the current sample. The size N of the observation window is chosen so that the operator output is maximum in the center of a street turn. The maxima in the operator output are used to segment the sequence into a list of street segments. The samples corresponding to the center of a turn are stored on an array of "turn coordinates".

In 702 the turn coordinates are grouped in clusters so that turns that belong to the same cluster are turns made on the same street intersection. An estimate of the coordinates of the street intersection center is obtained averaging the coordinates of the turn coordinates belonging to the same cluster. Using the street intersection coordinates, straight segments are further segmented into elementary street block segments and labeled with the "From" and "To" intersections.

In 703 correspondences between the street intersections computed above and the street intersections provided by a third party street maps database are computed. Each of the street intersections computed above is put in correspondence with the street intersection from the third party database that is the closest to it.

In 704 elementary street segments are put in correspondence with street segments from the third party street map database using the correspondences between the "From" and "To" intersections computed above. Each segment is labeled, using information from the third party database, with street name, orientation, and range of street numbers.

Note: the technique described above is particularly suitable for the 1-camera acquisition scheme. In the case of a 2-cameras acquisition scheme, the intersection coordinates would be detected using the intersection of straight segments rather than averaging the coordinates of the turns.

Generation of synthetic images database (Flow Chart 8,9, 10 and 10B)

Given video footage recorded from the side cameras and stored in the format of a DV file (Digital Video file) on a computer hard disk and the trajectory of the vehicle while acquiring such information, we want to compute the panoramic view like the one shown in **Figure 11**.
Since the viewing angle of the video camera is limited, a single video frame can only capture a small part of the street block (**Figure 12** is an example of video frame). One way to increase the field of view is to use a shorter focal but this method introduces image distortion. A second method would consist in increasing the distance between the camera and the buildings, but this is not always possible since the road is limited in width and there are other constructions on the opposite side of the street.
The method we propose is to synthesize the image that the camera would see if it were far away, using the whole sequence of image frames acquired.

For each street segment, synthetic views of both sides of the street are computed from the video sequence (801). In the current embodiment such images are computed at increments of 8 meters along the street segment. In 802 each synthetic image is store in a file in JPEG format with a file name that contains the following information:

- A numerical index that identifies the street segment
- the distance (in meters) of the location at which that image was computed from the origin of the street segment,
- the side of the street imaged

In 803 all the files are stored on magnetic storage.

Figure 5 is a pictorial description of the method used to create the synthetic panorama. Depicted in green are the successive positions of the real camera as the vehicle moves along the x axis direction. Depicted in red is the position of the virtual camera that sees the large view of the street block. The images on the top are three of the video frames extracted from the video sequence acquired by the camera. The image in the middle is the end product, the synthetic panorama. This image is created column by column, extracting each column from one of the video frames. To create column p_i in the panorama (the blue column in the figure), we have to find the video frame that was acquired when the video-camera was located at x_i . From the GPS reading we find at what exact time, t_i, the camera was in that location (see **Figure 4**). We then extract from the video sequence the video frame that was acquired at time t_i . From that frame we extract column with index (p_i / 720) * 720, where the first 720 is the total number of columns in the synthetic panorama and the second 720 is the number of columns in the video frame.

Virtual panoramas of the type shown in Figure 11 are then generated, extracting the appropriate pixel information from each one of the video frames acquired while driving through the city block. The distance between two adjacent panorama centers is 8 meters. Panorama images are stored in jpeg format, 720x120 pixels and about 15-20KByte in size.
At the same time close-ups and fish-eye views are also extracted from the video files. Both close-ups and fish-eye views are 360x240 pixels and about 13KBytes in size.

Close-up views are computed every 4 meters while fish-eye views are computed every 8 meters since the rate of change is much lower.

Flow charts 9 and 10 describe in detail the process of creating a synthetic panorama step by step:
In 901, given the arc length coordinate of the center of the panorama x_c, an array of evenly spaced arc length coordinates centered around x_c is computed.

In 902, using the map from UTC time to arc length (**Figure 4**) the array of UTC times corresponding to the array of arc length coordinates is computed.

In 903, using the time phase computed in the calibration step (see next section), UTC time is converted to video time.

In 904, for each video time, a column of RGB pixel values is extracted from one or more frames in the video sequence.

In 905, all the columns of RGB pixel values are stacked side by side to create the synthetic panorama bitmap which is then converted in a format more suitable for efficient storage (JPEG, GIF).

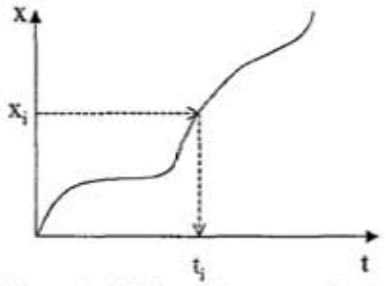


Figure 4: GPS reading x = x(time)

Flow charts 10 and 10B describe two possible ways of computing each column of RGB values.

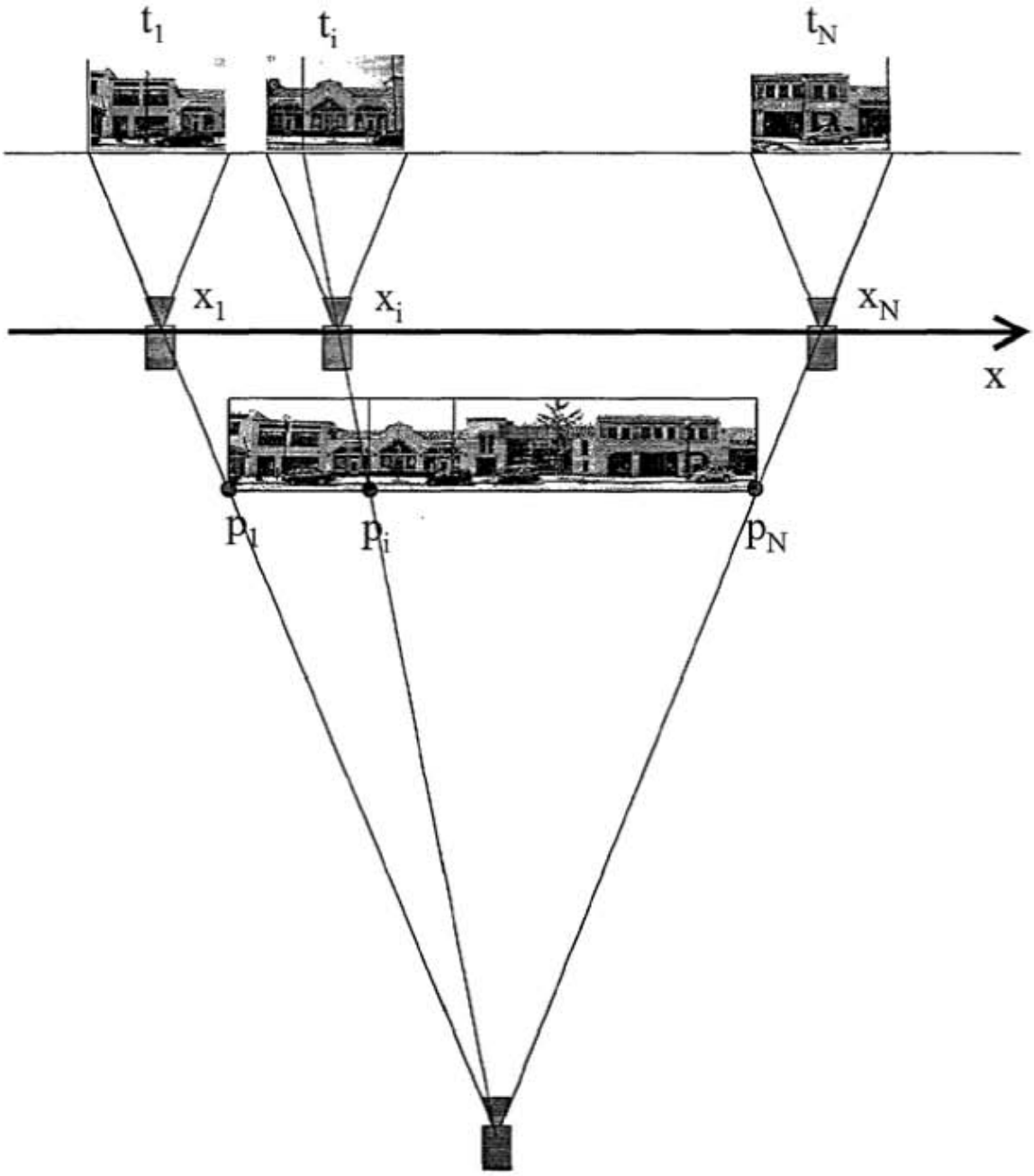


Figure 5: Pictorial description of the method used to create the synthetic panorama

Since the camera video sampling rate is finite, video frames are available only at finite times, multiple of the video sampling period of 1/60 sec (when using interlaced fields). Since a video frame at the desired time t_i is not available, we need to compute it from the available frames. A first order approximations (described in flow chart 10) is to use the closest video frame. This approximation is acceptable if the camera moves less than the distance between panorama columns in half of the sampling rate. The distance between two columns is 48m/720 = 0.066 cm, therefore the max speed is 0.066m/(1/120sec) = 8 m/sec which is about 20 miles/h.

In 1001 the field time which is the closest to the desired time is detected.

In 1002 the current video sequence frame is set to be the one acquired at that time.

In 1003 the column of RGB pixel values is extracted from the current frame buffer.

In case of higher camera speeds or lower sampling rates it will be necessary to reconstruct the frame at time t_i using the two or more adjacent video-frames (flow chart 10B). This is done computing optical flow on the video sequence. Optical flow is a vector field that assigns to each pixel in the image a 2-dimensional velocity vector. This allows to predict where the generic element pictured in pixel (i,j) in frame n will end-up being pictured in the next frame n+1.

Note 1. Distortion along the vertical direction; the method described above introduces a distortion along the vertical (y) direction due to the fact that we are not able to apply the same sampling technique that we use along the horizontal (x) direction. Since we copy an entire column of pixels as seen by one of the green cameras to a column in the panorama, the perspective view we obtain is not the correct one.
Figure 6 exemplifies this concept. Objects A and B appear to be the same size in the green camera, while they appear to have different sizes in the correct perspective view of the virtual red camera. When we use the column of image brightness of the green camera as the column of image brightness of the red camera, the objects in the synthetic view will appear distorted: objects close to the camera will appear taller than their real size and objects far away will appear shorter. In Figure 11 this effect is not very noticeable since all objects are located at approximately the same distance from the camera.

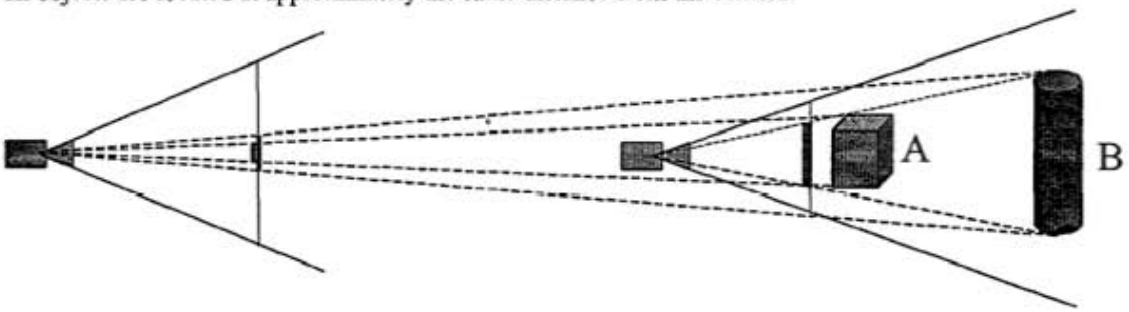


Figure 6

Figure 7 shows a panorama in which the distortion effect is more visible. The gray car closer to the camera appears to be taller then normal, the two cars stopped at the intersection appear with the right aspect ratio and the car and the condominium in the background appear shorter then their normal size.

Figure 7: Example of distortion along vertical direction

This type of distortion can be corrected if an estimate of the depth of each pixel in the panorama is available. Such depth (the distance from the camera) can be computed using optical flow. The aspect ratio of each pixel will be adjusted base on the distance of the object visualized in that pixel. One drawback of this solution is that gaps between pixels might be created (in **Figure 6** the red object occludes most of the blue object in the green camera view. In the red camera view parts of the blue object become visible but it's not possible to know what those parts look like since the green camera doesn't see them in that view (– however, the necessary information might be available in adjacent views where the red object does not occlude the blue one)).

One possible solution to this problem would consist in acquiring images from an array of two or more video cameras arranged along the vertical axis.

Note 2. Assumption of straight path (no turns, no bumps): The method of generating synthetic panoramas depicted in **Figure 5** relies on the assumption that the vehicle path is along a straight line. If this is not the case, then the choice of column to sample from a particular video frame will be incorrect. **Figure 8** shows two types of distortions that can occur. (1):At the left of the image, the vehicle undergoes a right-hand turn onto Colorado Blvd. This creates a distorted (bent) view of Colorado Blvd. (2): Near the first palm tree at Crate&Barel, the car passes over a bump. This tilts the camera, causing the extracted columns of the image to not be in vertical alignment with the pavement, and induces a wavy effect on the synthetic panorama. **Figure 9** shows the effect of changing lane. At the end of the Crocodile Café building, the car makes a slight leftward turn to change lanes. This causes the camera to pan left (opposite the traveling directions) and induces artifacts (mirror imaging) in the panorama. Notice the also the apparent change in height of the building due to the fact that the camera is closer after changing lanes.

Figure 8 Distortions due to turning onto a street (left side of image) and passing over a bump (near Crate&Barel palm tree).

Figure 9 Distortion due to changing lane (right side of Crocodile Café building).

The distortions due to bumps, lane change, deviation from a straight driving path can be corrected to some degree using optical flow to detect such situations and compensate for their effect.

The present method is a compromise between quality of the synthetic image and algorithmic complexity. Much more sophisticated and computationally intensive techniques for 3 dimensional reconstruction of the scene are available in the prior art. However, for the proposed application, the quality of the panoramic images is sufficient even in presence of some distortion.

Future embodiments of the invention could present video/image data in different formats. For example, rather than using a camera facing directly to the street side, a slightly forward (or backward)-looking camera could be used to provide a panoramic look up (or down) the street. Also, if sufficient cameras to cover all viewing directions are used (so as to provide 360 degrees of view) images (and synthetic panoramas) where the direction of view is user-controllable can be provided. The synthetic panoramic view can also be made into streaming video (by computing the panoramic view at increments of 30 cm or so), or even a streaming video with user-controllable viewing direction. As a last example, the synthetic panoramas could also be computed at several resolutions (moving the synthetic camera closer or further away from the street so as to see less or more of it) to provide the user with more levels of zooming in or out.

Sequences calibration (Flow charts 3,4,5 and 6)

For the production of high quality synthetic images, it is necessary to have an accurate synchronization of the video with the GPS position data. Figure 10 is an example where the GPS data is late by one second: at the second tree, the car has actually stopped moving and subsequent video frames are all the same. But the GPS data says we are moving, so we get a stretched tree (stationary video gets stretched along the path).

Immediately afterward, the car starts moving (video changes significantly every frame), but the GPS data thinks we are stopped (or moving slowly) so the tree gets squashed (a lot of video footage gets squished into a small change of position).

A post-processing method of accurately synchronizing video with GPS is described in flow chart 4:
In 401 we detect a landmark which is driven by in both directions (example, a tree in a lane divider). In 402 the time interval between the two passings of the landmark can be measured to within $1/30^{th}$ of a second in the video. In 403, using the GPS data, a function can be computed to determine the interval between successive passes of any point along the street segment. Since the two passages are in opposite directions, the function is monotonic, and thus an unambiguous and accurate determination of the GPS position (and time) of the landmark can be determined (404). Thus the time of passage in the video and the GPS data are known.

Flow chart 3 describes a method very similar to the previous one. It's also based on identifying a landmark that is driven by in both directions (301). It requires a search in the time phase space. The value of the time phase is initialized arbitrarily (302) and then modified (305) in order to minimize the measured distance between the two points on the vehicle trajectory that correspond to the time instants in the video sequence where the landmark is seen from the two sides of the street (303).

Synchronization of video and geo-location sequences during acquisition (flow chart 5):
Video and GPS can also be synchronized automatically during acquisition by determining the correspondence between the camcorder timestamp and GPS timestamp. Each GPS sample records the UTC (Universal Time Coordinate) of the measurement. When the camera begins recording, the camera clock time is read through the LANC interface. A UTC card on the PC correlates the camera time to UTC, and hence to the GPS samples.

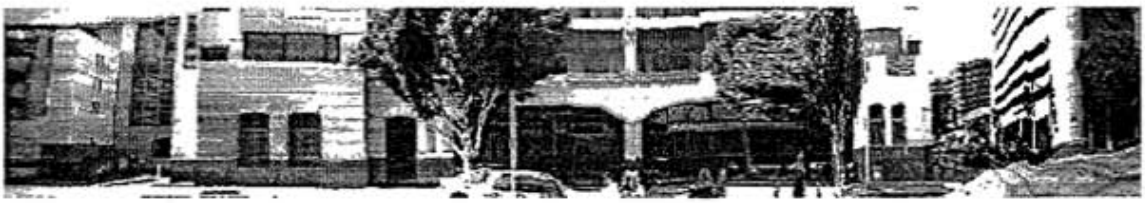


Figure 10: Image distortion due to inaccurate synchronization of video with GPS data

Flow chart 6 describes a fourth method for synchronizing the geo-location and the video sequences. This method requires the computation of optical flow of the bottom pixel row in the video sequence.
The choice of the bottom row is based on the assumption that the camera sees some of the curb or of the street pavement in that row. The average velocity for those pixels is directly proportional to the vehicle tangential velocity. Therefore, the time phase between the two sequences can be determined as the time delay that maximizes the alignment of local maxima and local minima between the average pixel velocity computed in 601 from the video sequence and the vehicle tangential velocity computed in 602 from the geo-location coordinates (603).

Generation of relational database

In order to allow efficient access to the data from a client, information regarding street segments is organized in the following Tables of a relational database:

Street Segments Table

Contains one entry per street segment. In the case of a 1-camera acquisition system, there are two vehicle trajectory segments that correspond to each street segment. The fields in each record include the following data:

- Street name
- Side of street with respect to main city street (N, S, E, or W). This parameter appears in front of the street name in an address.
- Street segment ID
- End-point coordinates of the third party city map segment corresponding to this segment

- Array of segment IDs of the segment adjacent to the segment start point ordered by direction (called the "From" address)
- Array of segment IDs of the segment adjacent to the segment end point ordered by direction (called the "To" address)
- The distance in meters of the start of the vehicle trajectory segment on each side of the street from the main city street (called Distance to Hub)
- The length in meters of the vehicle trajectory segment on each side of the street
- Offset in meters for each side of the street. Street numbers are assigned to properties based on their distance from the two main city streets (one runs N-S and one runs E-W). Sometimes there are deviations from this rule and these two offsets are used to account for those deviations.

Panorama Coordinates Table
This table contains one record per synthetic panorama contained in the panorama archive.
The fields of the record are:
- Segment ID of the street segment partially visualized in the image
- The side (even/odd) of the street segment visualized
- The distance from the segment origin of the center of the panorama in meters

Geo-location Look-up Table
In order to be able to find in an efficient manner the segment ID of the segment that is the closest to an arbitrary geo-location, a look-up table is built as follows: the city is divided in a regular grid of squared blocks. The size of the block is chosen so that the average number of street segments crossing a block is sufficiently small (for example, 10). For each block, the list of segment IDs of the street segments crossing the block is determined.
The database table contains one record per list element of each block. The fields of each record are:
- Block label (for example, block center coordinates)
- Segment ID

In order to practice this invention, there can be many alternative ways of organizing the information in the database and the one described above is what we used for the preferred embodiment.

In addition, one or more databases containing various type of information indexed by city address can be used to annotate the synthetic panoramas. For example, in the present embodiment a business listing was used to annotate the panoramas with information about the displayed businesses (name of the business, street address, phone number, type of activity, hyperlink to a web site).

2. Street Browsing tool

Description

The principal commodity is a visual database of streets in the US. It contains images of street views both as a lower resolution panorama of a side street block (see Figure 11) and as higher resolution close-up views of both a specific property (see Figure 12) and of the street view. The video database is indexed both by street address and by geographical coordinates (latitude and longitude). Each elementary unit is a jpeg image with average size 18Kbytes for panoramas and 13Kbytes for single frames allowing for fast download and low latency.
In the technology section we describe the modalities with which we will build such a database and the cost associated.

Figure 11: Example of virtual panorama image

The Scout Tool (see Figure 13) is the user interface that allows accessing the video database in several modalities:

- **address mode**: the user types in an address of the location he would like to visualize
- **geo-location mode**: the user clicks on the map the location he/she would like to visualize
- **browse-mode**: from the current location, the user can click on the directional buttons to literally drive through the city streets. The "switch view" button allows seeing the other side of the street.

At any time, the user can click on a specific property on the panorama to obtain a higher resolution close-up view.
Since each horizontal coordinate on the panorama is automatically indexed with the corresponding street name and address, a table with "yellow pages" type of information can be shown on top of the panorama image including a hyperlink to the web site of that property.

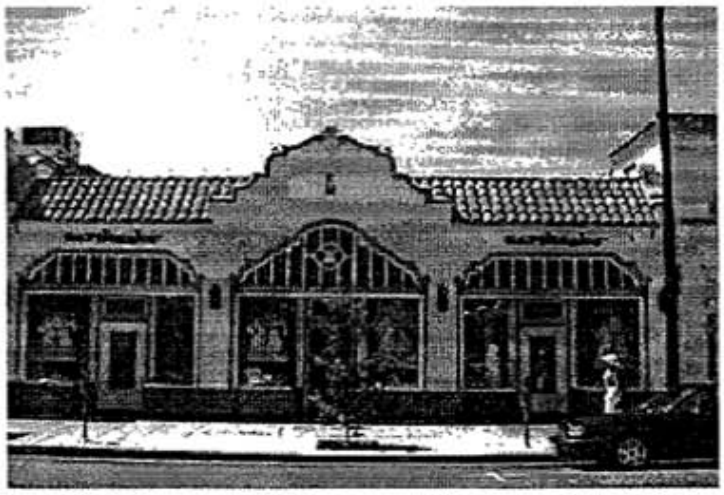


Figure 12: Example of close-up view

The alpha version of the client-side of the software is implemented in VBScript and can run inside Internet Explorer version 4 or higher. The user is not required to download any special plug-in in order to use the tool.

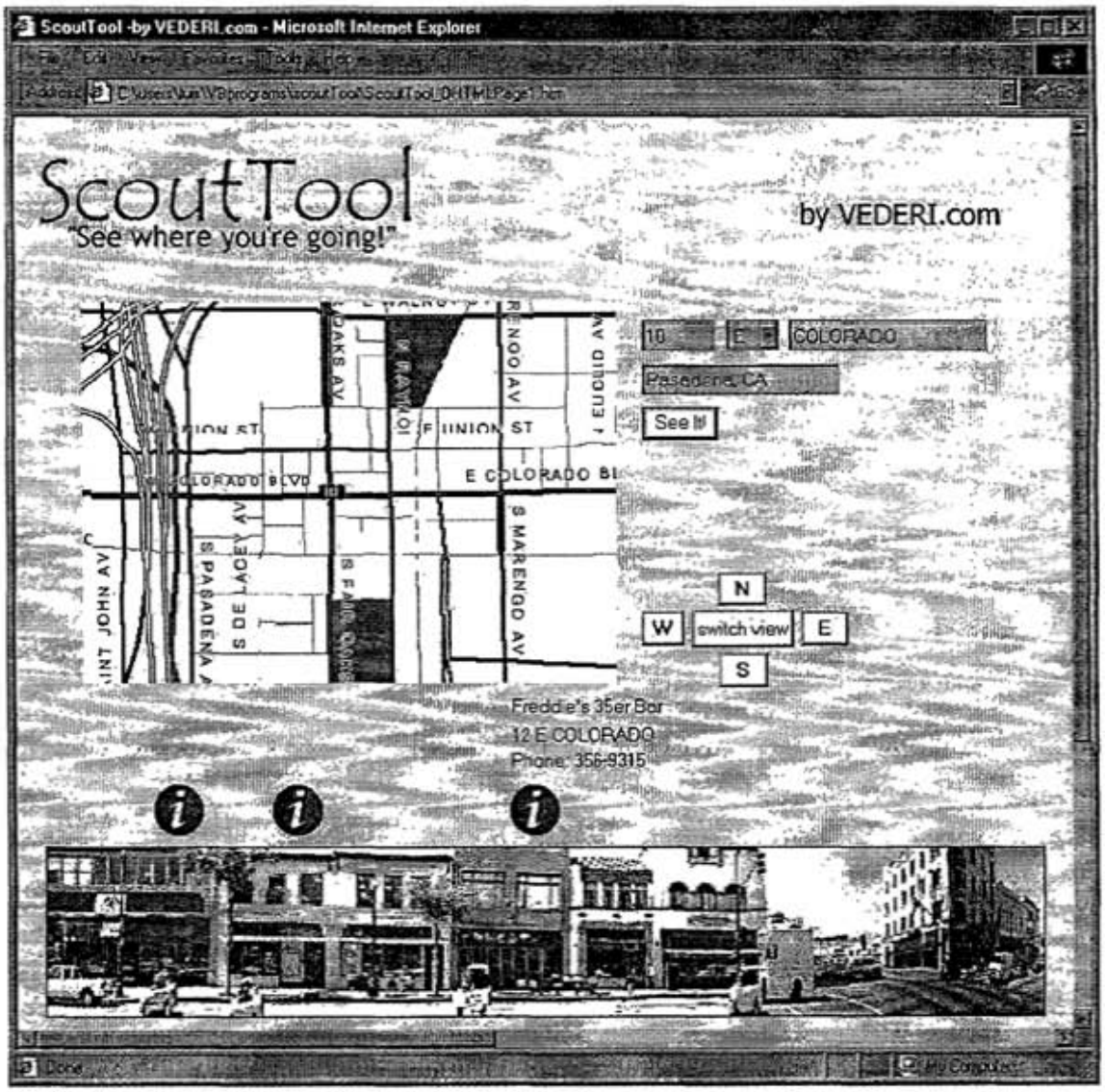


Figure 13: Vederi Scout Tool

Implementation of address look-up

Flowchart 13 describes the method of accessing the video database given a desired street address.
In 1301, the user types in the desired address in the ScoutTool's street number, street name, city and state text input fields.
In 1302, when the user presses the "See It", the address is sent to the server.
In 1303, the server performs a query on the database of street segments to find the street segment that corresponds to the desired address. The query searches for street segments that match the desired street name (and city and state), and for which the street number lies within the "From" and "To" street numbers. The search returns the appropriate segment index number, as well as the relative position of the desired number along the street segment. This relative position is computed as the distance in meters from the start of the street segment (the "From" street number) based on the fact that street numbers increment by two every 12.5 feet (every 3.81meters).
In 1304, the query result (segment index number and distance along segment) is sent back to the client, along with a new map of the location if necessary. The map is generated by third-party software such as

provided by Navigation Technologies or Etak. The software is capable of generating a street map of a location given a street address with which to center the map. The map generated can be much larger than the portion viewable by the user on the client machine so that the user can scout around without having to download a new map very often.

In 1305, the client requests from the server information about the new street segment to be viewed. This information includes: the geographic coordinates of the start and stop of the street segment (4 entries because they may be slightly different for the even and odd side of the street), the "To" and "From" street numbers on each side of the street, the length in meters of the segment, the distance of the start of the street segment on each side of the street from the 'Hub' (the Hub is a reference location from which street numbers are defined. For example, in Pasadena California, the Hub is the intersection of Colorado Boulevard and Fair Oaks Avenue. Street numbers start from that intersection, and 'North' or 'South' are relative to Colorado Blvd, East-West are relative to Fair Oaks Avenue.), the positions along the segment at which synthetic panoramas have been computed, annotation information (such as business listing information) for properties visible on that segment.

In 1306, the client requests from the server the synthetic panorama containing a view of the desired address. The client chooses amongst the list of computed synthetic panoramas for that segment, the one with the closest distance to the desired address (which was transmitted to the client in step 1303), and then downloads that specific panorama from the server.

In 1307, the ScoutTool display is updated with all the new information. This includes

- displaying the new panorama,
- displaying the new map (or centering the new one on the current address being viewed),
- updating the car Icon and view direction Icon on the map to graphically represent the current viewing location and direction,
- updating the possible navigation buttons in accord with the possible directions of motion from the new current position, and
- displaying icons or business logos or annotations above the panoramic image at the appropriate horizontal position corresponding to the property to which the information pertains.

Implementation of geo-location look-up

Flowchart 14 describes the method of visualizing a location by clicking on a location on a map.

In step 1401, the user clicks on a location on the map displayed by the client-side application (the ScoutTool).

In step 1402, the map coordinates are converted from screen coordinates (location of mouse click) to relative Mercator coordinates and sent to the server. The conversion from mouse click screen coordinates to relative Mercator coordinates is done by:

- retrieving the position of the mouse click relative to the screen origin
- adding the map display offset to compute the screen coordinates relative to the map origin
- converting from map screen coordinates to relative Mercator coordinates by the appropriate scaling and offset function used in the creation of the map.

In step 1403, the server runs a query to find the street segment that is closest to the desired geo-location. This is done by first selecting the relevant segment block. The two-dimensional block index is computed by rounding the requested Mercator coordinates to the nearest hundred. The segment block information (which was generated during the creation of the database) holds a list of segment indexes. The desired location is compared with each of the segments listed in order to find the closest one. Since each segment has a start and stop coordinate (the "From" and "To" coordinates) and is represented as a line segment, the closest segment is found by

- choosing the segment from the list for which the perpendicular distance from the segment to the desired location is smallest, and for which
- the projection of the vector from the "From" coordinate to the desired location onto the line containing the street segment lies within the "From" coordinate and the "To" coordinate.

In step 1404 the index of the closest street segment is returned to the client. Additionally, a new map centered on the desired location can be transmitted to the client if the previous map did not cover an extent large enough to allow the display centered on the desired location.

In step 1405 the client requests from the server the database record pertaining to the desired street segment. (same as step 1305)

In step 1406 the client determines which of the segment's associated panoramas shows the desired location and retrieves the panoramic image from the server. The selection of the panorama is done by:

- converting the Mercator coordinates to a distance along the street segment (since the length of the street segment is known, and the starting and ending Mercator coordinates are known, a position along the street segment can be calculated)
- finding the associated panorama with the closest distance to the desired street distance.

In step 1407 the ScoutTool display is updated with the new information (same as step 1307)

Implementation of navigation control

Flowchart 15 describes the method of navigation by moving the current position incrementally in a chosen direction.

In Step 1501 the user clicks on one of the enabled direction buttons to move East, West, North or South.

In Step 1502 the current segment index and current panorama are updated accordingly. The current position is incremented 8meters to the next available panorama in the street segment. If the end of the street is reached, or movement onto the other street at an intersection is requested, a new segment must be retrieved.

In step 1503 the client retrieves the information for the new street segment moved to. The required segment index is know because the segment info record has fields that denote which segments connect to it on either end in the North, South, East, and West directions.

In step 1504 the client retrieves the image for the new current position. This is either the next panorama on the same segment, or the first (or last) panorama on the new adjacent segment.

In step 1505 the ScoutTool display is updated with the new information (same as step 1307)

Implementation of selection of close-up image

Flowchart 16 describes the method for selecting and displaying a close-up view

In step 1601 the user clicks on a particular location in the synthetic image to be viewed in close-up mode.

In step 1602 the relative position of the location clicked with respect to the start of the segment is computed. This is done by

- retrieving the mouse coordinates on the screen of the location clicked
- converting the mouse coordinates to panoramic image coordinates (adding the coordinate offset of the image)
- computing the distance of the location clicked from the location in the center of the panoramic image according to the formula pos = (coordinate – image_width/2)*64_meters/image_width. (Each panorama represents 64 meters of street)
- computing the distance of the location clicked from the start of the street segment by adding the relative distance from the center of the panorama to the distance of the panorama from the start of the street segment

In step 1603 the closest close-up image is computed by rounding the computed distance from the start of the segment to the nearest 4 meters.

In step 1604 the closest close-up image is retrieved specifying to the server the street segment index and the close-up distance (rounded to the nearest 4 meters).

In step 1605 a new browser window is opened which displays the close-up view

Implementation of display of business listings

Flowchart 17 describes the method of displaying business listing information (or any other location-based annotation)

In step 1701, for each business info listing associated with the current street segment, it is checked whether or not the business is visible in the current panorama. This is done by converting the street number in the business info into a horizontal pixel coordinate in the panoramic image. This is in turn is done by:

- subtracting the street segment "From" address from the street number, multiplying by 4.81m and dividing by 2 to obtain the distance of the location from the start of the segment (in meters).

- Subtracting the distance of the panorama along the street segment from the distance of the location from the start of the segment, dividing by 64 meters and multiplying by length of the panorama in pixels (720).

If the resulting pixel position of the location is within 0 to 720 pixels, it is visible within the panorama.
In step 1702, if the business info lies within the panorama, a business info icon is created and placed at that horizontal position above the panorama. The index of the business info entry is also stored in a list of created business logos.
In step 1703 the business info icon is programmed so that when the mouse is passed above it, it will trigger the display of the available business information above the icon.
In step 1704 the business info icon is programmed so that if the associated business info includes a website, clicking on the business info icon will open another browser and display that website.

NOTES:

The description describes an embodiment of the invention where the database only covered the city of Pasadena, California. The same invention can easily be adapted to cover larger geographical regions, such as the entire United States, by including record fields of city and state into the database and modifying all the queries to include that information.

The current embodiment of the invention is a navigable image database of the city of Pasadena, California. The invention could also be used to create a navigable image database of locales other than urban street scenes where it would be useful for a user to visually navigate throughout the environment. For example, using an electric golf cart as a mobile platform, a database of a museum could be created. The end user could then visually navigate throughout the museum, zoom in on works of art, and read associated annotation information. Other examples are the creation of a navigable database of Disney World, the San Diego Zoo, or Yosemite National Park.

Appendix – VBScript for ScoutTool

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML><HEAD>

<SCRIPT LANGUAGE="vbscript">

Option Explicit
' we travel 48 meters along car path, but that corresponds to a larger view at the store front
' used to compute streetNumber given position in panorama
' and to access the desired video frame when panorama is clicked
Const PanoramaLengthAtStoreFront = 64 'As Double = 64

Const MAP_TOP  = 120 'the coordinates of the map on the webpage
Const MAP_LEFT  = 20

Const MAP_CENTER_X  = 207
Const MAP_CENTER_Y  = 135

Const CAR_ICON_POS_X  = 198 '= MAP_CENTER_X - 9
Const CAR_ICON_POS_Y  = 126 '= MAP_CENTER_Y - 9
Const DOT_ICON_DELTA  = 8

Const MAX_LOGO = 39'As Integer = 9 'from 0..3 logos available for use
Dim logoBusinessInd(39)' (MAX_LOGO) 'As Integer ' indexes the current business listing using n'th
logo

Dim curMapClip(2)  ' remember the top and left pixel of the clipped region of the map

Const VEDERI_IP  = "24.41.30.120"

Const ButtonOffColor = "#cfeff6"
Const ButtonOnColor = "#ffffcc"

'Dim segmentInfo 'As SegmentInfoType

  Dim segmentInfoSegmentIndex 'As Integer
  Dim segmentInfostreetName 'As String
  Dim segmentInfoSideOfHub 'As String ' one of "E", "W", "N", "S"

  Dim segmentInfoFromMapCoordGeo(2) 'As Double ' in order Latitude, Longitude
  Dim segmentInfoFromMapCoordMercat(2) 'As Double ' mercator coordinates, in order X, Y

  Dim segmentInfoToMapCoordGeo(2) 'As Double ' in order Latitude, Longitude
  Dim segmentInfoToMapCoordMercat(2) 'As Double

  Dim segmentInfoFromSegment(4) 'As Integer ' in order E, N, W, S
  Dim segmentInfoToSegment(4) 'As Integer ' in order E, N, W, S

  Dim segmentInfoDistanceFromHub(2) 'As Double 'in order Even, Odd side
  Dim segmentInfoLength(2) 'As Double ' in order Even, Odd side
  Dim segmentInfoOffset(2) 'As Double ' even, odd side
  Dim segmentInfoOffsetFlag(2) 'As Double ' even, odd.  0 - unchanged, 1 - hard constraint (manually
adjusted)
                                    ' 2 - impossible to determine (no address available)
                                    ' 3 - automatically estimated (and could be inaccurate)
  Dim segmentInfoNumPanoramas(2) 'As Integer ' even, odd side
  Dim segmentInfoNumFrames(2) 'As Integer
  Dim segmentInfoFrameSequence(2) 'As Integer

  'LG000914
  dim segmentInfoNumBusinesses(2) 'As Integer ' number of business listings available for 0 - even,
1 - odd street side

'No Type definition in VBscript, unfortunately...
'Type BusinessInfoType
'    Name As String
'    Address As String
'    Phone As String
'    URL As String
'End Type
```

```
'Dim businessInfo() As BusinessInfoType ' businessInfo(side=[0,1],busNumber=[0..NumBusinesses(side)-
1])
dim businessInfoName(2,39)' (MAX_LOGO)
dim businessInfoAddress(2,39)' (MAX_LOGO)
dim businessInfoPhone(2,39)' (MAX_LOGO)
dim businessInfoURL(2,39)' (MAX_LOGO)

Dim curStreetSide  ' 0 - even side, 1 - odd side
Dim curPanIndex  ' index to which is the panorama being displayed

Dim panoramaDist(2,100)  ' (2,NumPans) first index is 0 for even side, 1 for odd
                  ' list of the distances (in cm) of the panoramas from start of segment

Private Function convertENWStoNum(ByVal ENSWdir )
    'to facilitate indexing data
    Dim num 'As Integer
    Select Case ENSWdir
        Case "E", "e":
            num = 0
        Case "N", "n":
            num = 1
        Case "W", "w":
            num = 2
        Case "S", "s":
            num = 3
    End Select
    convertENWStoNum = num
End Function

Private Function convertNumToENWS(ByVal num )
    Select Case num
        Case 0:
            convertNumToENWS = "E"
        Case 1:
            convertNumToENWS = "N"
        Case 2:
            convertNumToENWS = "W"
        Case 3:
            convertNumToENWS = "S"
    End Select
End Function

Private Function RetrieveNewSegmentInfo(ByVal segIndex )
    ' connect to remote database and get entry for new segment
    ' this means retrieving segmentInfo and panoramaDist()
    ' if the segment doesn't exist, don't download anything, and return false
    Dim sqlResult
    Dim sqlSA
    If segIndex = 0 Then
        RetrieveNewSegmentInfo = False
    Else
            'sqlResult =
Inet1.Execute("http://24.41.30.120/ScoutTool/GetsegmentInfo.asp?SegmentIndex=" & segIndex,"GET")
        sqlResult =
Inet1.OpenURL("http://24.41.30.120/ScoutTool/GetSegmentAndBusinessInfo.asp?SegmentIndex=" &
segIndex)
        'check if connection worked, if not, give up
        if sqlResult <> "" then

        sqlSA = Split(sqlResult, Chr(10))
        if not isnumeric(sqlSA(0)) then
               Msgbox "Inet failed and did not return new segment info."
               exit function
        end if

        'Now fill in the variable fields...
        segmentInfoSegmentIndex = CLng(sqlSA(0))
        segmentInfostreetName = sqlSA(1)
        segmentInfoSideOfHub = sqlSA(2)

        segmentInfoFromMapCoordGeo(0) = Cdbl(sqlSA(3))
        segmentInfoFromMapCoordGeo(1) = Cdbl(sqlSA(4))
        segmentInfoFromMapCoordMercat(0) = Cdbl(sqlSA(5))
        segmentInfoFromMapCoordMercat(1) = Cdbl(sqlSA(6))
```

```
        segmentInfoToMapCoordGeo(0) = Cdbl(sqlSA(7))
        segmentInfoToMapCoordGeo(1) = Cdbl(sqlSA(8))
        segmentInfoToMapCoordMercat(0) = Cdbl(sqlSA(9))
        segmentInfoToMapCoordMercat(1) = Cdbl(sqlSA(10))

        segmentInfoFromSegment(0) = Clng(sqlSA(11))
        segmentInfoFromSegment(1) = Clng(sqlSA(12))
        segmentInfoFromSegment(2) = Clng(sqlSA(13))
        segmentInfoFromSegment(3) = Clng(sqlSA(14))

        segmentInfoToSegment(0) = Clng(sqlSA(15))
        segmentInfoToSegment(1) = Clng(sqlSA(16))
        segmentInfoToSegment(2) = Clng(sqlSA(17))
        segmentInfoToSegment(3) = Clng(sqlSA(18))

        segmentInfoDistanceFromHub(0) = Cdbl(sqlSA(19))
        segmentInfoDistanceFromHub(1) = Cdbl(sqlSA(20))

        segmentInfoLength(0) = Cdbl(sqlSA(21))
        segmentInfoLength(1) = Cdbl(sqlSA(22))

        segmentInfoOffset(0) = Cdbl(sqlSA(23))
        segmentInfoOffset(1) = Cdbl(sqlSA(24))
        segmentInfoOffsetFlag(0) = CInt(sqlSA(25))
        segmentInfoOffsetFlag(1) = CInt(sqlSA(26))

        segmentInfoNumPanoramas(0) = CInt(sqlSA(27))
        segmentInfoNumPanoramas(1) = CInt(sqlSA(28))
        segmentInfoNumFrames(0) = CInt(sqlSA(29))
        segmentInfoNumFrames(1) = CInt(sqlSA(30))
        segmentInfoFrameSequence(0) = CInt(sqlSA(31))
        segmentInfoFrameSequence(1) = CInt(sqlSA(32))
        '*********************************************************************

        'Now read in panoramaDist data
        'read in even and odd sides separately
        'use FrameSequence = -1 to indicate that side of street was not scanned

        Dim maxCount
        If segmentInfoNumPanoramas(0) > segmentInfoNumPanoramas(1) Then
            maxCount = segmentInfoNumPanoramas(0)
        Else
            maxCount = segmentInfoNumPanoramas(1)
        End If
        'VBscript doesn't like redim so comment out and start with large dim
        'ReDim panoramaDist(2, maxCount) 'As Long

        'Don't use separate call because this .asp hangs for some reason.
        'even side
        'sqlResult =
Inet1.OpenURL("http://24.41.30.120/ScoutTool/GetPanoramaCoords.asp?SegmentIndex=" & segIndex &
"&EvenOdd=0")
        'sqlSA = Split(sqlResult, " ")
        'Dim i 'As Integer
        'For i = 0 To UBound(sqlSA) - 1
        '    panoramaDist(0, i) = sqlSA(i)
        'Next ' ' i

        Dim i 'As Integer
        For i = 0 To segmentInfoNumPanoramas(0) - 1
            panoramaDist(0, i) = sqlSA(i + 33)
        Next ' '

        'odd side
        'sqlResult =
Inet1.OpenURL("http://24.41.30.120/ScoutTool/GetPanoramaCoords.asp?SegmentIndex=" & segIndex &
"&EvenOdd=1")
        'sqlSA = Split(sqlResult, " ")
        ''Dim i As Integer
        'For i = 0 To UBound(sqlSA) - 1
        '    panoramaDist(1, i) = sqlSA(i)
        'Next ' ' i
```

```
        'Dim i As Integer
        For i = 0 To segmentInfoNumPanoramas(1) - 1
            panoramaDist(1, i) = sqlSA(i + 33 + segmentInfoNumPanoramas(0))
        Next '  '  i


        ' read business info
        ' Don't know how many there are on even and odd side off hand
        ' and they will arrive unsorted
        Dim eCount, oCount 'As Integer ' count how many even, odd listings there are
        eCount = 0
        oCount = 0

        Dim sqlOffset 'As Integer
        sqlOffset = 33 + SegmentInfoNumPanoramas(0) + SegmentInfoNumPanoramas(1)

        Dim tCount 'As Integer ' the total is known because we know how many lines
                        ' of text we received through Inet1
        tCount = (UBound(sqlSA) - sqlOffset) / 4 'four lines of Inet1 per business listing

               'No redim in VBscript, have already defined max size of businesssInfo variables
        'ReDim businessInfo(2, tCount) As BusinessInfoType

        For i = 1 To tCount
            'check if it's even or odd
            'order of lines are  bAddress, bName, bPhone, bURL
            If sqlSA(sqlOffset + 4 * (i - 1)) Mod 2 = 0 Then
                'Add another even side listing
                businessInfoName(0, eCount) = sqlSA(sqlOffset + 4 * (i - 1) + 1)
                businessInfoAddress(0, eCount) = sqlSA(sqlOffset + 4 * (i - 1))
                businessInfoPhone(0, eCount) = sqlSA(sqlOffset + 4 * (i - 1) + 2)
                businessInfoURL(0, eCount) = sqlSA(sqlOffset + 4 * (i - 1) + 3)
                eCount = eCount + 1
            Else
                'add another odd side listing
                businessInfoName(1, oCount) = sqlSA(sqlOffset + 4 * (i - 1) + 1)
                businessInfoAddress(1, oCount) = sqlSA(sqlOffset + 4 * (i - 1))
                businessInfoPhone(1, oCount) = sqlSA(sqlOffset + 4 * (i - 1) + 2)
                businessInfoURL(1, oCount) = sqlSA(sqlOffset + 4 * (i - 1) + 3)
                oCount = oCount + 1
            End If
        Next 'i

        SegmentInfoNumBusinesses(0) = eCount
        SegmentInfoNumBusinesses(1) = oCount

        RetrieveNewSegmentInfo = True

        'NOTE: HACK
        '      check if curStreetSide has any panoramas. If not, we have to switch
        '      to the other side, or else we get errors accessing panoramaDist()
        If segmentInfoNumPanoramas(curStreetSide) = 0 Then
            curStreetSide = 1 - curStreetSide
        End If

        Else ' if sqlResult <> ""
            MsgBox "Inet1 returned the empty string"
        End If
    End If
End Function

Private Function btnMoveEast_onclick() 'As Boolean
    If btnMoveEast.Value <> "" Then
        'button is enabled, so move!
        Navigate (0)
    End If
End Function

Private Function btnMoveNorth_onclick() 'As Boolean
    If btnMoveNorth.Value <> "" Then
        Navigate (1)
    End If
End Function
```

```
Private Function btnMoveWest_onclick() 'As Boolean
    If btnMoveWest <> "" Then
        Navigate (2)
    End If
End Function

Private Function btnMoveSouth_onclick() 'As Boolean
    If btnMoveSouth <> "" Then
        Navigate (3)
    End If
End Function

Private Function btnSeeIt_onclick() 'As Boolean
    'search for the address
    Dim sqlResult 'As String
    Dim sqlSA 'As String
    'replace street name spaces with underscores for CGI request
    Dim streetName 'As String
    streetName = Replace(txtStreetName.Value, " ", "_")
    sqlResult = Inet1.OpenURL("http://" & VEDERI_IP &
"/ScoutTool/Address2Location.exe?StreetNumber=" & txtStreetNumber.Value & "&StreetName=" &
streetName & "&SideofHub=" & convertNumToENWS(selectENWS.selectedIndex))
    ' returns segmentIndex, distance along segment in meters
    If sqlResult <> "" Then
        sqlSA = Split(sqlResult, " ")
        If IsNumeric(sqlSA(0)) Then
            Dim newSeg 'As Integer
            newSeg = CLng(sqlSA(0))
            If newSeg <> 0 Then
                Dim requestDist 'As Double
                requestDist = Cdbl(sqlSA(1))

                'retrieve the new segment, and find the closest panorama
                If RetrieveNewSegmentInfo(newSeg) Then

                    curStreetSide = CInt(txtStreetNumber.Value) Mod 2
                    dim panAndSide 'need to return two values
                    panAndSide = FindClosestPanorama(curStreetSide, requestDist * 100)
                    curPanIndex = CLng(panAndSide(0))
                    curStreetSide = CInt(panAndSide(1))

                    'display it
                    UpdateNavigationButtons
                    DisplayCurrentPanorama
                End If
            Else
                'address does not exist, or not available in our database
                MsgBox "That address is not visualizable", vbOKOnly, "Address not Found"
            End If
        End If
    End If
    'txtDebug.Value = Inet1.OpenURL("http://news.bbc.co.uk/low/english/sci/tech/default.stm")
End Function

Private Function btnSwitchView_onclick() 'As Boolean
    'check if panoramas exist on the other side of the street
    'don't switch if there aren't any!
    If segmentInfoNumPanoramas(1 - curStreetSide) <> 0 Then
        curStreetSide = 1 - curStreetSide

        DisplayCurrentPanorama
    End If
End Function

Private Sub DHTMLPage_Load()
    'NOTE: Not sure if this is where to put initialization
    '      ??? What is the equivalent of Form Load in DHTML

    'turn off scroll bars since it screws up mouse position
    'this gives an error
    'document.parentWindow.window.scrollbars.visibility = "False"

    'Load in vederi logo
    imgVederiLogo.src = "http://" & VEDERI_IP & "/ScoutTool/vederiLogo.gif"
```

```
'Load in map
imgMap.src = "http://" & VEDERI_IP & "/ScoutTool/pasadenaMap.gif"
'resize the image to the true size'
imgMap.Style.Height = 1273 '1638
imgMap.Style.Width = 1638
'clip it! order of params is  top, right, bottom, left
'No, can't clip, or IE will crash!!!
'imgMap.Style.clip = "rect(205px 515px 475px 100px)" 'clip size of 415x270
'reposition on screen.  make the (0,0) be window pixel 20,20
imgMap.Style.left = -100 + MAP_LEFT 'shift from 20 by clip.left
imgMap.Style.top = -205 + MAP_TOP ' shift from 20 by clip.top

curMapClip(0) = 100
curMapClip(1) = 205

'set the cursor to something clickable
imgMap.Style.cursor = "hand"


'load in map icons
imgCarIcon.src = "http://" & VEDERI_IP & "/ScoutTool/carNS.gif"
imgCarIcon.Style.Height = 20
imgCarIcon.Style.Width = 20
imgCarIcon.Style.left = MAP_LEFT + CAR_ICON_POS_X
imgCarIcon.Style.top = MAP_TOP + CAR_ICON_POS_Y

imgDotIcon.src = "http://" & VEDERI_IP & "/ScoutTool/dotIcon.gif"
imgDotIcon.Style.Height = 19
imgDotIcon.Style.Width = 19
imgDotIcon.Style.left = MAP_LEFT + CAR_ICON_POS_X + DOT_ICON_DELTA
imgDotIcon.Style.top = MAP_TOP + CAR_ICON_POS_Y

'setup txtBusinessInfo info window
txtBusinessInfoName.Style.backgroundColor = "#cfeff6"
txtBusinessInfoAddress.Style.backgroundColor = "#cfeff6"
txtBusinessInfoPhone.Style.backgroundColor = "#cfeff6"

txtBusinessInfoName.Style.border = 0
txtBusinessInfoAddress.Style.border = 0
txtBusinessInfoPhone.Style.border = 0


'set up business logo buttons (the markers which you can click for more info)
imgLogo0.src = "http://" & VEDERI_IP & "/ScoutTool/infosign.gif"
imgLogo1.src = "http://" & VEDERI_IP & "/ScoutTool/infosign.gif"
imgLogo2.src = "http://" & VEDERI_IP & "/ScoutTool/infosign.gif"
imgLogo3.src = "http://" & VEDERI_IP & "/ScoutTool/infosign.gif"
imgLogo4.src = "http://" & VEDERI_IP & "/ScoutTool/infosign.gif"
imgLogo5.src = "http://" & VEDERI_IP & "/ScoutTool/infosign.gif"
imgLogo6.src = "http://" & VEDERI_IP & "/ScoutTool/infosign.gif"
imgLogo7.src = "http://" & VEDERI_IP & "/ScoutTool/infosign.gif"
imgLogo8.src = "http://" & VEDERI_IP & "/ScoutTool/infosign.gif"
imgLogo9.src = "http://" & VEDERI_IP & "/ScoutTool/infosign.gif"

curStreetSide = 0
DisableNS
RetrieveNewSegmentInfo (141)

curPanIndex = 1

'scale panorama to true size
'imgPanorama.Style.Width = 720
'imgPanorama.Style.Height = 120

'set the cursor to something clickable on panorama
imgPanorama.style.cursor = "hand"
DisplayCurrentPanorama

'setup street address
selectENWS.selectedIndex = 2
txtStreetName.Value = segmentInfostreetName
txtStreetNumber.Value = 101
```

```
End Sub

Private Function imgLogo_onclick() 'As Boolean
    'clicking on busines logo/info opens new browser to their website

    'Dim closeupWindow 'As HTMLWindow2
    'Set closeupWindow = Document.parentWindow.window.open("http://www.vision.caltech.edu/luis",
"closeupWindow")
    Document.parentWindow.window.open "http://www.vision.caltech.edu/luis", "businessLinkWindow"
End Function

Private Function imgMap_onclick() 'As Boolean

    'click on map means jump to the nearest segment & panorama
    Dim mousePos(2) 'As Integer
    mousePos(0) = Document.parentwindow.window.event.x - imgMap.offsetLeft - curMapClip(0)
    mousePos(1) = Document.parentWindow.window.event.y - imgMap.offsetTop - curMapClip(1)

    'compute carIcon current position in mercator coordinates
    Dim curMerc(2) 'As Double
    curMerc(0) = (segmentInfoToMapCoordMercat(0) - segmentInfoFromMapCoordMercat(0)) * (curPanIndex
/ (segmentInfoNumPanoramas(curStreetSide) + 1)) + segmentInfoFromMapCoordMercat(0)
    curMerc(1) = (segmentInfoToMapCoordMercat(1) - segmentInfoFromMapCoordMercat(1)) * (curPanIndex
/ (segmentInfoNumPanoramas(curStreetSide) + 1)) + segmentInfoFromMapCoordMercat(1)

    'compute mercator coordinates of mouseclick position
    curMerc(0) = curMerc(0) + (mousePos(0) - MAP_CENTER_X) / 1571600
    curMerc(1) = curMerc(1) + (mousePos(1) - MAP_CENTER_Y) / -1569000

    txtDebug.Value = "Map pos: " & curMerc(0) & " , " & curMerc(1)

    'query for closest segment and panorama
    Dim sqlResult 'As String
    Dim sqlSA 'As String
    sqlResult = Inet1.OpenURL("http://" & VEDERI_IP & "/ScoutTool/Coords2Segment.exe?XCoord=" &
curMerc(0) & "&YCoord=" & curMerc(1))
    'returns segmentIndex, street side, and percentage along length of segment
    If sqlResult <> "" Then
        sqlSA = Split(sqlResult, " ")
        If IsNumeric(sqlSA(0)) Then
            If sqlSA(0) <> 0 Then
                Dim newSeg 'As Integer
                newSeg = CLng(sqlSA(0))
                Dim newStreetSide 'As Integer
                newStreetSide = CInt(sqlSA(1))
                Dim requestDist 'As Double
                requestDist = Cdbl(sqlSA(2))

                'retrieve the new segment, and find the closest panorama
                If RetrieveNewSegmentInfo(newSeg) Then
                                    dim panAndSide 'need to return two values
                    panAndSide = FindClosestPanorama(newStreetSide, 100 * requestDist *
segmentInfoLength(newStreetSide))
                    curPanIndex = CLng(panAndSide(0))
                    curStreetSide = CInt(panAndSide(1))

                    UpdateNavigationButtons
                    DisplayCurrentPanorama
                End If
            Else
                'address does not exist, or not available in cur database
                MsgBox "That address is not visualizable", vbOKOnly, "Address not Found"
            End If
        End If
    End If

End Function

Private Function imgPanorama_onclick() 'As Boolean
    'click on panorama means show the close-up view at that position

    'open window right away so people don't wonder what's going on
    Dim browserOptions 'As String
    Dim closeupBrowser 'As IHTMLWindow2
```

```
    browserOptions = "toolbar=No, location=No, directories=No, menubar=No, scrollbars=No, width=380,
height=260"
    Set closeupBrowser = Document.parentWindow.window.open("http://" & VEDERI_IP &
"/ScoutTool/downloadingCloseup.gif", "closeupWindow", browserOptions)
    'closeupBrowser.focus


    'compute the desired frame index
    Dim frameIndex 'As Integer
    Dim mousePos 'As Integer
    mousePos = Document.parentwindow.window.event.x- imgPanorama.offsetLeft

    ' relative position in pan * (48meters/pan.width) + panorama dist from start of segment , divide
by 1/3m and round
    ' see notes EDB 08/25/00 page 4

    'LG000920: save frames to disk ahead of time - don't need to access video.
    '          compute a frame every 4 meters from start of segment
    Dim framePos 'As Double
    framePos = ComputePanoramaParity() * (mousePos - imgPanorama.Width / 2) * _
        (PanoramaLengthAtStoreFront / imgPanorama.Width) + _
        panoramaDist(curStreetSide, curPanIndex) / 100
    'round to nearest 4 meters
    Dim framePosRounded 'As Long
    framePosRounded = 4 * Round(framePos / 4)

    If framePosRounded < 0 Then
        framePosRounded = 0
    End If
    If framePosRounded > SegmentInfoLength(curStreetSide) Then
        framePosRounded = Int(SegmentInfoLength(curStreetSide) / 4) * 4
    End If

    browserOptions = "toolbar=No, location=No, directories=No, menubar=No," & _
        "scrollbars=No, width=380, height=260"

    closeupBrowser.focus
    Set closeupBrowser = Document.parentWindow.window.open( _
        "http://" & VEDERI_IP & "/Frames/Seg" & SegmentInfoSegmentIndex & "_" _
        & curStreetSide & "_D" & framePosRounded * 100 & "_Fr.jpg", _
        "closeupWindow", browserOptions)
    'closeupBrowser.open _
    '    "http://" & VEDERI_IP & "/Frames/Seg" & SegmentInfoSegmentIndex & "_" _
    '    & curStreetSide & "_D" & framePosRounded * 100 & "_Fr.jpg", _
    '    "closeupWindow", browserOptions
    'closeupBrowser.Document.title = "Vederi.com - Close-up view"

End Function

Private Function ComputePanoramaParity() 'As Integer
    'when computing close-up frame index and frame time, need to know if moving
    'mouse left on panorama goes towards "From" address or "To" address.
    Select Case curStreetSide
        Case 0:
            'looking at the even side of the street
            Select Case segmentInfoSideOfHub
                Case "W", "w", "S", "s":
                    ComputePanoramaParity = 1
                Case "N", "n", "E", "e":
                    ComputePanoramaParity = -1
            End Select
        Case 1:
            'looking at the odd side of the street
            Select Case segmentInfoSideOfHub
                Case "W", "w", "S", "s":
                    ComputePanoramaParity = -1
                Case "N", "n", "E", "e":
                    ComputePanoramaParity = 1
            End Select
    End Select
End Function

Private Sub DisplayCurrentPanorama()
```

```
    imgPanorama.src = "http://24.41.30.120/Panoramas/Seg" & segmentInfoSegmentIndex & "_" &
curStreetSide & "_D" & panoramaDist(curStreetSide, curPanIndex) & ".jpg"

    'Hide any previous business info
    HideBusinessInfo

    'Hide all the logos
    HideBusinessLogos

    'Go through list of businesses on street side and determine which ones are on panorama
    Dim curLogo 'As Integer
    curLogo = 0

    Dim bInd 'As Integer
    For bInd = 0 To SegmentInfoNumBusinesses(curStreetSide) - 1
        'compute position on panorama
        Dim delta 'As Double 'distance (in meters) of street number from center of panorama
        delta = (businessInfoAddress(curStreetSide, bInd) + curStreetSide) * 3.81 / 2.0 -
SegmentInfoDistanceFromHub(curStreetSide) - panoramaDist(curStreetSide, curPanIndex) / 100 +
SegmentInfoOffset(curStreetSide)
        Dim logoPos 'As Double
        logoPos = delta * imgPanorama.Width / PanoramaLengthAtStoreFront * ComputePanoramaParity() +
imgPanorama.Width / 2

        If logoPos > 0 And logoPos < imgPanorama.Width Then
            'add a business logo
            AddBusinessLogo bInd, curLogo, logoPos
            curLogo = curLogo + 1
        End If
    Next 'bInd

    UpdateMap
    txtDebug.Value = "http://24.41.30.120/Panoramas/Seg" & segmentInfoSegmentIndex & "_" &
curStreetSide & "_D" & panoramaDist(curStreetSide, curPanIndex) & ".jpg"

End Sub

Private Sub DisableNS()
    btnMoveNorth.Style.visibility = "hidden"
    btnMoveSouth.Style.visibility = "hidden"
End Sub

Private Sub EnableNS()
    btnMoveNorth.Style.visibility = "visible"
    btnMoveSouth.Style.visibility = "visible"
End Sub

Private Sub DisableEW()
    btnMoveEast.Style.visibility = "hidden"
    btnMoveWest.Style.visibility = "hidden"
End Sub

Private Sub EnableEW()
    btnMoveEast.Style.visibility = "visible"
    btnMoveWest.Style.visibility = "visible"
End Sub

Private Sub Navigate(direction)
    'main routine to move given a N,S,E,W button click
    'input : direction to move in : 0 = East, 1 = North, 2 = West, 3 = South

    'compute "relative direction of motion"
    Dim relDir 'As Integer
    relDir = direction - convertENWStoNum(segmentInfoSideOfHub)
    Select Case relDir
        Case 0:
            'going toward "To" address
            MoveTowardsToAddress direction
        Case 1, -3:
            'turning left on "To" address, or right on "From" address
            If curPanIndex > (segmentInfoNumPanoramas(curStreetSide) / 2) Then
                TurnAtToAddress direction
            Else
                TurnAtFromAddress direction
```

```
            End If
        Case 2, -2:
            'going towards "From" address
            MoveTowardsFromAddress direction
        Case 3, -1:
            'turning right on "To" address, or left on "From" address
            If curPanIndex > (segmentInfoNumPanoramas(curStreetSide) / 2) Then
                TurnAtToAddress direction
            Else
                TurnAtFromAddress direction
            End If
    End Select
End Sub

Private Sub MoveTowardsToAddress(ByVal direction )
    Dim newPanIndex 'As Integer
    newPanIndex = curPanIndex + 1
    If newPanIndex >= segmentInfoNumPanoramas(curStreetSide) Then
        'need to move to next segment
        If RetrieveNewSegmentInfo(segmentInfoToSegment(direction)) Then
            'usually, we should be at the "From" address, unless we've just crossed the hub
            If convertENWStoNum(segmentInfoSideOfHub) = direction Then
                curPanIndex = 0
            Else
                curPanIndex = segmentInfoNumPanoramas(curStreetSide) - 1
            End If
        End If
    Else
        curPanIndex = newPanIndex
    End If
    UpdateNavigationButtons
    DisplayCurrentPanorama
End Sub

Private Sub MoveTowardsFromAddress(ByVal direction )
    Dim newPanIndex 'As Integer
    newPanIndex = curPanIndex - 1
    If newPanIndex < 0 Then
        'need to move to next segment
        If RetrieveNewSegmentInfo(segmentInfoFromSegment(direction)) Then
            'usually, we should be at the "To" address, unless we've just crossed the hub
            If convertENWStoNum(segmentInfoSideOfHub) = direction Then
                curPanIndex = 0
            Else
                curPanIndex = segmentInfoNumPanoramas(curStreetSide) - 1
            End If
        End If
    Else
        curPanIndex = newPanIndex
    End If
    UpdateNavigationButtons
    DisplayCurrentPanorama

End Sub

Private Sub TurnAtToAddress(ByVal direction )
    If RetrieveNewSegmentInfo(segmentInfoToSegment(direction)) Then
        'NOTE: may want to redefine curStreetSide so that we see the side we were on.
        '      (as if we were driving a car)
        If convertENWStoNum(segmentInfoSideOfHub) = direction Then
            curPanIndex = 0
        Else
            curPanIndex = segmentInfoNumPanoramas(curStreetSide) - 1
        End If
    End If
    UpdateNavigationButtons
    DisplayCurrentPanorama
End Sub

Private Sub TurnAtFromAddress(ByVal direction )
    If RetrieveNewSegmentInfo(segmentInfoFromSegment(direction)) Then
        'NOTE: may want to redefine curStreetSide so that we see the side we were on.
        '      (as if we were driving a car)
        If convertENWStoNum(segmentInfoSideOfHub) = direction Then
```

```
            curPanIndex = 0
        Else
            curPanIndex = segmentInfoNumPanoramas(curStreetSide) - 1
        End If
    End If
    UpdateNavigationButtons
    DisplayCurrentPanorama
End Sub

Private Sub UpdateNavigationButtons()
    If curPanIndex = 0 Then
        'check which segments we can go to from the From Address
        EnableSegments segmentInfoFromSegment
    ElseIf curPanIndex = segmentInfoNumPanoramas(curStreetSide) - 1 Then
        'check which segments we can go to from the To Address
        EnableSegments segmentInfoToSegment
    Else
        'we can only travel along the segment
        Select Case segmentInfoSideOfHub
            Case "N", "n", "S", "s":
                EnableNS
                DisableEW
            Case "E", "e", "W", "w":
                EnableEW
                DisableNS
        End Select
    End If
End Sub

Private Sub EnableSegments(segs() )
    If segs(0) <> 0 Then
        btnMoveEast.Style.visibility = "visible"
    Else
        btnMoveEast.Style.visibility = "hidden"
    End If

    If segs(1) <> 0 Then
        btnMoveNorth.Style.visibility = "visible"
    Else
        btnMoveNorth.Style.visibility = "hidden"
    End If
    If segs(2) <> 0 Then
        btnMoveWest.Style.visibility = "visible"
    Else
        btnMoveWest.Style.visibility = "hidden"
    End If
    If segs(3) <> 0 Then
        btnMoveSouth.Style.visibility = "visible"
    Else
        btnMoveSouth.Style.visibility = "hidden"
    End If
End Sub
Private Sub UpdateMap()
    'compute our current position in mercator coordinates
    Dim curMerc(2) 'As Double
    curMerc(0) = (segmentInfoToMapCoordMercat(0) - segmentInfoFromMapCoordMercat(0)) * (curPanIndex
/ (segmentInfoNumPanoramas(curStreetSide) + 1)) + segmentInfoFromMapCoordMercat(0)
    curMerc(1) = (segmentInfoToMapCoordMercat(1) - segmentInfoFromMapCoordMercat(1)) * (curPanIndex
/ (segmentInfoNumPanoramas(curStreetSide) + 1)) + segmentInfoFromMapCoordMercat(1)

    'compute image coordinates given mercator coordinates
    Dim imgCoords(2) 'As Integer
    imgCoords(0) = Round(1571600 * curMerc(0) + 869.73)
    imgCoords(1) = Round(-1569000 * curMerc(1) + 346.17)

    'clip and shift map to center on current location
    Dim top, right, bottom, left 'As Integer
    top = imgCoords(1) - MAP_CENTER_Y
    right = imgCoords(0) + MAP_CENTER_X
    left = imgCoords(0) - MAP_CENTER_X
    bottom = imgCoords(1) + MAP_CENTER_Y

    'clip it! order of params is  top, right, bottom, left
    imgMap.Style.clip = "rect(" & top & "px " & right & "px " & bottom & "px " & left & "px)"
```

```
    curMapClip(0) = left
    curMapClip(1) = top

    'reposition on screen.  make the (0,0) be window pixel 20,20
    imgMap.Style.left = -left + MAP_LEFT 'shift from 20 by clip.left
    imgMap.Style.top = -top + MAP_TOP ' shift from 20 by clip.top

    'adjust car and dot icons
    Select Case segmentInfoSideOfHub
        Case "E", "e", "W", "w":
            imgCarIcon.src = "http://" & VEDERI_IP & "/ScoutTool/carEW.gif"
            imgDotIcon.Style.left = MAP_LEFT + CAR_ICON_POS_X
            imgDotIcon.Style.top = MAP_TOP + CAR_ICON_POS_Y - (2 * curStreetSide - 1) *
DOT_ICON_DELTA
        Case "N", "n", "S", "s":
            imgCarIcon.src = "http://" & VEDERI_IP & "/ScoutTool/carNS.gif"
            imgDotIcon.Style.left = MAP_LEFT + CAR_ICON_POS_X - (2 * curStreetSide - 1) *
DOT_ICON_DELTA
            imgDotIcon.Style.top = MAP_TOP + CAR_ICON_POS_Y
    End Select

    'Update location info on webpage
    txtStreetName.Value = segmentInfostreetName
    selectENWS.selectedIndex = convertENWStoNum(segmentInfoSideOfHub)
    'compute approximate street address
    Dim newNumber 'As Integer
    newNumber = 4 * Round((segmentInfoDistanceFromHub(curStreetSide) + panoramaDist(curStreetSide,
curPanIndex) / 100 - segmentInfoOffset(curStreetSide)) / 3.81 / 2 - 0.5 * curStreetSide) +
curStreetSide
    txtStreetNumber.Value = newNumber
End Sub

Private Function FindClosestPanorama(ByRef requestedStreetSide , ByVal requestDist ) 'As Integer
    'find the closest panorama on the side of the street
    ' ALSO returns an updated requestedStreetSide, in case no panoramas on original side
    '(first, check if there are any panoramas on the side of the street)
    If segmentInfoNumPanoramas(requestedStreetSide) = 0 Then
        'no panoramas on that side, switch to the other
        '(the other will have panoramas for sure, or else the segement
        ' wouldn't be in the database)
        requestedStreetSide = 1 - requestedStreetSide
        'NOTE this hack for correcting street number will have to go...
        txtStreetNumber.Value = txtStreetNumber.Value + 1
    End If
    'now find closest panorama
    Dim panInd, bestInd 'As Integer
    Dim bestDist 'As Double
    bestDist = 99999
    bestInd = -1
    For panInd = 0 To segmentInfoNumPanoramas(requestedStreetSide) - 1
        If Abs(requestDist - panoramaDist(requestedStreetSide, panInd)) < bestDist Then
            bestInd = panInd
            bestDist = Abs(requestDist - panoramaDist(requestedStreetSide, panInd))
        End If
    Next '  '  panInd
    dim panAndSide(2)
    panAndSide(0) = bestInd
    panAndSide(1) = requestedStreetSide
    FindClosestPanorama = panAndSide
End Function

'Private Sub AddBusinessLogo(ByVal businessInd As Integer, ByVal curLogo As Integer, logoPosition As
Double)
Private Sub AddBusinessLogo(businessInd , curLogo , logoPosition )
    If curLogo <= MAX_LOGO Then
        logoBusinessInd(curLogo) = businessInd
        Select Case curLogo
            Case 0:
                imgLogo0.Title = businessInfoURL(curStreetSide, businessInd)
                imgLogo0.Style.left = logoPosition + imgPanorama.offsetLeft
                imgLogo0.Style.visibility = "visible"
            Case 1:
                imgLogo1.Title = businessInfoURL(curStreetSide, businessInd)
                imgLogo1.Style.left = logoPosition + imgPanorama.offsetLeft
```

```
                imgLogo1.Style.visibility = "visible"
            Case 2:
                imgLogo2.Title = businessInfoURL(curStreetSide, businessInd)
                imgLogo2.Style.left = logoPosition + imgPanorama.offsetLeft
                imgLogo2.Style.visibility = "visible"
            Case 3:
                imgLogo3.Title = businessInfoURL(curStreetSide, businessInd)
                imgLogo3.Style.left = logoPosition + imgPanorama.offsetLeft
                imgLogo3.Style.visibility = "visible"
            Case 4:
                imgLogo4.Title = businessInfoURL(curStreetSide, businessInd)
                imgLogo4.Style.left = logoPosition + imgPanorama.offsetLeft
                imgLogo4.Style.visibility = "visible"
            Case 5:
                imgLogo5.Title = businessInfoURL(curStreetSide, businessInd)
                imgLogo5.Style.left = logoPosition + imgPanorama.offsetLeft
                imgLogo5.Style.visibility = "visible"
            Case 6:
                imgLogo6.Title = businessInfoURL(curStreetSide, businessInd)
                imgLogo6.Style.left = logoPosition + imgPanorama.offsetLeft
                imgLogo6.Style.visibility = "visible"
            Case 7:
                imgLogo7.Title = businessInfoURL(curStreetSide, businessInd)
                imgLogo7.Style.left = logoPosition + imgPanorama.offsetLeft
                imgLogo7.Style.visibility = "visible"
            Case 8:
                imgLogo8.Title = businessInfoURL(curStreetSide, businessInd)
                imgLogo8.Style.left = logoPosition + imgPanorama.offsetLeft
                imgLogo8.Style.visibility = "visible"
            Case 9:
                imgLogo9.Title = businessInfoURL(curStreetSide, businessInd)
                imgLogo9.Style.left = logoPosition + imgPanorama.offsetLeft
                imgLogo9.Style.visibility = "visible"
        End Select
    End If
End Sub

Private Sub HideBusinessInfo()
    txtBusinessInfoName.Style.visibility = "hidden"
    txtBusinessInfoAddress.Style.visibility = "hidden"
    txtBusinessInfoPhone.Style.visibility = "hidden"
End Sub

Private Sub HideBusinessLogos()
    imgLogo0.Style.visibility = "hidden"
    imgLogo1.Style.visibility = "hidden"
    imgLogo2.Style.visibility = "hidden"
    imgLogo3.Style.visibility = "hidden"
    imgLogo4.Style.visibility = "hidden"
    imgLogo5.Style.visibility = "hidden"
    imgLogo6.Style.visibility = "hidden"
    imgLogo7.Style.visibility = "hidden"
    imgLogo8.Style.visibility = "hidden"
    imgLogo9.Style.visibility = "hidden"
End Sub

Private Function imgLogo0_onclick() 'As Boolean
    'clicking on busines logo/info opens new browser to their website
    If imgLogo0.Title = "" Or imgLogo0.Title = " " Then
        MsgBox "Web link information not available.", , "URL not available"
    Else
        Document.parentWindow.window.open "http://" & imgLogo0.Title, "businessLinkWindow"
    End If
End Function

Private Function imgLogo0_onclick() 'As Boolean
    'clicking on busines logo/info opens new browser to their website
    If imgLogo0.Title = "" Or imgLogo0.Title = " " Then
        MsgBox "Web link information not available.", , "URL not available"
    Else
        Document.parentWindow.window.open "http://" & imgLogo0.Title, "businessLinkWindow"
    End If
End Function
```

```
Private Function imgLogo1_onclick() 'As Boolean
    'clicking on busines logo/info opens new browser to their website
    If imgLogo1.Title = "" Or imgLogo1.Title = " " Then
        MsgBox "Web link information not available.", , "URL not available"
    Else
        Document.parentWindow.window.open "http://" & imgLogo1.Title, "businessLinkWindow"
    End If
End Function

Private Function imgLogo2_onclick() 'As Boolean
    'clicking on busines logo/info opens new browser to their website
    If imgLogo2.Title = "" Or imgLogo2.Title = " " Then
        MsgBox "Web link information not available.", , "URL not available"
    Else
        Document.parentWindow.window.open "http://" & imgLogo2.Title, "businessLinkWindow"
    End If
End Function

Private Function imgLogo3_onclick() 'As Boolean
    'clicking on busines logo/info opens new browser to their website
    If imgLogo3.Title = "" Or imgLogo3.Title = " " Then
        MsgBox "Web link information not available.", , "URL not available"
    Else
        Document.parentWindow.window.open "http://" & imgLogo3.Title, "businessLinkWindow"
    End If
End Function

Private Function imgLogo3_onclick() 'As Boolean
    'clicking on busines logo/info opens new browser to their website
    If imgLogo3.Title = "" Or imgLogo3.Title = " " Then
        MsgBox "Web link information not available.", , "URL not available"
    Else
        Document.parentWindow.window.open "http://" & imgLogo3.Title, "businessLinkWindow"
    End If
End Function

Private Function imgLogo4_onclick() 'As Boolean
    'clicking on busines logo/info opens new browser to their website
    If imgLogo4.Title = "" Or imgLogo4.Title = " " Then
        MsgBox "Web link information not available.", , "URL not available"
    Else
        Document.parentWindow.window.open "http://" & imgLogo4.Title, "businessLinkWindow"
    End If
End Function

Private Function imgLogo5_onclick() 'As Boolean
    'clicking on busines logo/info opens new browser to their website
    If imgLogo5.Title = "" Or imgLogo5.Title = " " Then
        MsgBox "Web link information not available.", , "URL not available"
    Else
        Document.parentWindow.window.open "http://" & imgLogo5.Title, "businessLinkWindow"
    End If
End Function

Private Function imgLogo6_onclick() 'As Boolean
    'clicking on busines logo/info opens new browser to their website
    If imgLogo6.Title = "" Or imgLogo6.Title = " " Then
        MsgBox "Web link information not available.", , "URL not available"
    Else
        Document.parentWindow.window.open "http://" & imgLogo6.Title, "businessLinkWindow"
    End If
End Function

Private Function imgLogo7_onclick() 'As Boolean
    'clicking on busines logo/info opens new browser to their website
    If imgLogo7.Title = "" Or imgLogo7.Title = " " Then
        MsgBox "Web link information not available.", , "URL not available"
    Else
        Document.parentWindow.window.open "http://" & imgLogo7Title, "businessLinkWindow"
    End If
End Function

Private Function imgLogo8_onclick() 'As Boolean
    'clicking on busines logo/info opens new browser to their website
```

```
    If imgLogo8.Title = "" Or imgLogo8.Title = " " Then
        MsgBox "Web link information not available.", , "URL not available"
    Else
        Document.parentWindow.window.open "http://" & imgLogo8.Title, "businessLinkWindow"
    End If
End Function

Private Function imgLogo9_onclick() 'As Boolean
    'clicking on busines logo/info opens new browser to their website
    If imgLogo9.Title = "" Or imgLogo9.Title = " " Then
        MsgBox "Web link information not available.", , "URL not available"
    Else
        Document.parentWindow.window.open "http://" & imgLogo9.Title, "businessLinkWindow"
    End If
End Function

Private Sub imgLogo0_onmouseover()
    'Display the business info
    txtBusinessInfoName.Value = businessInfoName(curStreetSide, logoBusinessInd(0))
    txtBusinessInfoAddress.Value = businessInfoAddress(curStreetSide, logoBusinessInd(0)) & " " &
convertNumToENWS(selectENWS.selectedIndex) & " " & txtStreetName.Value
    txtBusinessInfoPhone.Value = "Phone: " & businessInfoPhone(curStreetSide, logoBusinessInd(0))

    DisplayBusinessInfo (imgLogo0.offsetLeft)
End Sub

Private Sub imgLogo1_onmouseover()
    'Display the business info
    txtBusinessInfoName.Value = businessInfoName(curStreetSide, logoBusinessInd(1))
    txtBusinessInfoAddress.Value = businessInfoAddress(curStreetSide, logoBusinessInd(1)) & " " &
convertNumToENWS(selectENWS.selectedIndex) & " " & txtStreetName.Value
    txtBusinessInfoPhone.Value = "Phone: " & businessInfoPhone(curStreetSide, logoBusinessInd(1))

    DisplayBusinessInfo (imgLogo1.offsetLeft)
End Sub

Private Sub imgLogo2_onmouseover()
    'Display the business info
    txtBusinessInfoName.Value = businessInfoName(curStreetSide, logoBusinessInd(2))
    txtBusinessInfoAddress.Value = businessInfoAddress(curStreetSide, logoBusinessInd(2)) & " " &
convertNumToENWS(selectENWS.selectedIndex) & " " & txtStreetName.Value
    txtBusinessInfoPhone.Value = "Phone: " & businessInfoPhone(curStreetSide, logoBusinessInd(2))

    DisplayBusinessInfo (imgLogo2.offsetLeft)
End Sub

Private Sub imgLogo3_onmouseover()
    'Display the business info
    txtBusinessInfoName.Value = businessInfoName(curStreetSide, logoBusinessInd(3))
    txtBusinessInfoAddress.Value = businessInfoAddress(curStreetSide, logoBusinessInd(3)) & " " &
convertNumToENWS(selectENWS.selectedIndex) & " " & txtStreetName.Value
    txtBusinessInfoPhone.Value = "Phone: " & businessInfoPhone(curStreetSide, logoBusinessInd(3))

    DisplayBusinessInfo (imgLogo3.offsetLeft)
End Sub

Private Sub imgLogo4_onmouseover()
    'Display the business info
    txtBusinessInfoName.Value = businessInfoName(curStreetSide, logoBusinessInd(4))
    txtBusinessInfoAddress.Value = businessInfoAddress(curStreetSide, logoBusinessInd(4)) & " " &
convertNumToENWS(selectENWS.selectedIndex) & " " & txtStreetName.Value
    txtBusinessInfoPhone.Value = "Phone: " & businessInfoPhone(curStreetSide, logoBusinessInd(4))

    DisplayBusinessInfo (imgLogo4.offsetLeft)
End Sub

Private Sub imgLogo5_onmouseover()
    'Display the business info
    txtBusinessInfoName.Value = businessInfoName(curStreetSide, logoBusinessInd(5))
    txtBusinessInfoAddress.Value = businessInfoAddress(curStreetSide, logoBusinessInd(5)) & " " &
convertNumToENWS(selectENWS.selectedIndex) & " " & txtStreetName.Value
    txtBusinessInfoPhone.Value = "Phone: " & businessInfoPhone(curStreetSide, logoBusinessInd(5))

    DisplayBusinessInfo (imgLogo5.offsetLeft)
```

```
End Sub

Private Sub imgLogo6_onmouseover()
    'Display the business info
    txtBusinessInfoName.Value = businessInfoName(curStreetSide, logoBusinessInd(6))
    txtBusinessInfoAddress.Value = businessInfoAddress(curStreetSide, logoBusinessInd(6)) & " " &
convertNumToENWS(selectENWS.selectedIndex) & " " & txtStreetName.Value
    txtBusinessInfoPhone.Value = "Phone: " & businessInfoPhone(curStreetSide, logoBusinessInd(6))

    DisplayBusinessInfo (imgLogo6.offsetLeft)
End Sub

Private Sub imgLogo7_onmouseover()
    'Display the business info
    txtBusinessInfoName.Value = businessInfoName(curStreetSide, logoBusinessInd(7))
    txtBusinessInfoAddress.Value = businessInfoAddress(curStreetSide, logoBusinessInd(7)) & " " &
convertNumToENWS(selectENWS.selectedIndex) & " " & txtStreetName.Value
    txtBusinessInfoPhone.Value = "Phone: " & businessInfoPhone(curStreetSide, logoBusinessInd(7))

    DisplayBusinessInfo (imgLogo7.offsetLeft)
End Sub

Private Sub imgLogo8_onmouseover()
    'Display the business info
    txtBusinessInfoName.Value = businessInfoName(curStreetSide, logoBusinessInd(8))
    txtBusinessInfoAddress.Value = businessInfoAddress(curStreetSide, logoBusinessInd(8)) & " " &
convertNumToENWS(selectENWS.selectedIndex) & " " & txtStreetName.Value
    txtBusinessInfoPhone.Value = "Phone: " & businessInfoPhone(curStreetSide, logoBusinessInd(8))

    DisplayBusinessInfo (imgLogo8.offsetLeft)
End Sub

Private Sub imgLogo9_onmouseover()
    'Display the business info
    txtBusinessInfoName.Value = businessInfoName(curStreetSide, logoBusinessInd(9))
    txtBusinessInfoAddress.Value = businessInfoAddress(curStreetSide, logoBusinessInd(9)) & " " &
convertNumToENWS(selectENWS.selectedIndex) & " " & txtStreetName.Value
    txtBusinessInfoPhone.Value = "Phone: " & businessInfoPhone(curStreetSide, logoBusinessInd(9))

    DisplayBusinessInfo (imgLogo9.offsetLeft)
End Sub

Private Sub DisplayBusinessInfo(ByVal offsetLeft )' As Integer)
    If offsetLeft < imgPanorama.offsetLeft Then
        offsetLeft = imgPanorama.offsetLeft
    End If
    If offsetLeft > imgPanorama.offsetLeft + imgPanorama.Width - 200 Then
        offsetLeft = imgPanorama.offsetLeft + imgPanorama.Width - 200
    End If

    txtBusinessInfoName.Style.left = offsetLeft
    txtBusinessInfoName.Style.visibility = "visible"
    txtBusinessInfoAddress.Style.left = offsetLeft
    txtBusinessInfoAddress.Style.visibility = "visible"
    txtBusinessInfoPhone.Style.left = offsetLeft
    txtBusinessInfoPhone.Style.visibility = "visible"

End Sub

</SCRIPT>
</HEAD>


<BODY onload="DHTMLPage_Load" BGCOLOR="#CFEFF6">

<IMG id=imgPanorama name=imgPanorama src="" style="HEIGHT: 120px; LEFT: 18px; POSITION: absolute;
TOP: 506px; WIDTH: 720px; Z-INDEX: 118" height=120 width=720 border=1>

<IMG alt="Vederi Map" border=0 height=964 hspace=0 id=imgMap name=imgMap src="about:blank"
style="HEIGHT: 289px; LEFT: 12px; POSITION: absolute; TOP: 107px; WIDTH: 416px; Z-INDEX: 100"
useMap="" width=415>

<TEXTAREA cols=25 id=txtDebug name=TextArea1 rows=4 style="HEIGHT: 114px; LEFT: 187px; POSITION:
absolute; TOP: 750px; WIDTH: 185px; Z-INDEX: 101">TextArea</TEXTAREA>
```

```
<IMG id=imgLogo0 name=imgLogo0 src="" style="HEIGHT: 24px; LEFT: 125px; POSITION: absolute; TOP:
470px; WIDTH: 24px; Z-INDEX: 102">
<IMG id=imgLogo1 name=imgLogo1 src="" style="HEIGHT: 24px; LEFT: 125px; POSITION: absolute; TOP:
470px; WIDTH: 24px; Z-INDEX: 102">
<IMG id=imgLogo2 name=imgLogo2 src="" style="HEIGHT: 24px; LEFT: 125px; POSITION: absolute; TOP:
470px; WIDTH: 24px; Z-INDEX: 102">
<IMG id=imgLogo3 name=imgLogo3 src="" style="HEIGHT: 24px; LEFT: 125px; POSITION: absolute; TOP:
470px; WIDTH: 24px; Z-INDEX: 102">
<IMG id=imgLogo4 name=imgLogo4 src="" style="HEIGHT: 24px; LEFT: 125px; POSITION: absolute; TOP:
470px; WIDTH: 24px; Z-INDEX: 102">
<IMG id=imgLogo5 name=imgLogo5 src="" style="HEIGHT: 24px; LEFT: 125px; POSITION: absolute; TOP:
470px; WIDTH: 24px; Z-INDEX: 102">
<IMG id=imgLogo6 name=imgLogo6 src="" style="HEIGHT: 24px; LEFT: 125px; POSITION: absolute; TOP:
470px; WIDTH: 24px; Z-INDEX: 102">
<IMG id=imgLogo7 name=imgLogo7 src="" style="HEIGHT: 24px; LEFT: 125px; POSITION: absolute; TOP:
470px; WIDTH: 24px; Z-INDEX: 102">
<IMG id=imgLogo8 name=imgLogo8 src="" style="HEIGHT: 24px; LEFT: 125px; POSITION: absolute; TOP:
470px; WIDTH: 24px; Z-INDEX: 102">
<IMG id=imgLogo9 name=imgLogo9 src="" style="HEIGHT: 24px; LEFT: 125px; POSITION: absolute; TOP:
470px; WIDTH: 24px; Z-INDEX: 102">

<INPUT id=txtStreetNumber name=txtStreetNumber style="HEIGHT: 22px; LEFT: 465px; POSITION: absolute;
TOP: 138px; WIDTH: 53px; Z-INDEX: 116" value=Number>
<SELECT id=selectENWS name=selectENWS style="HEIGHT: 22px; LEFT: 525px; POSITION: absolute; TOP:
138px; WIDTH: 39px; Z-INDEX: 115" value="Select1"><OPTION value =E>E</OPTION><OPTION selected
value=N>N</OPTION><OPTION value=W>W</OPTION><OPTION value=S>S</OPTION></SELECT>
<INPUT id=txtStreetName name=txtCityState style="LEFT: 570px; POSITION: absolute; TOP: 138px; Z-
INDEX: 109" value="Street Address">
<INPUT id=txtCityState name=txtCityState style="LEFT: 466px; POSITION: absolute; TOP: 170px; Z-
INDEX: 110" value="Pasadena, CA">

<INPUT id=btnSeeIt name=btnSeeIt style="LEFT: 465px; POSITION: absolute; TOP: 207px; Z-INDEX: 111;
BACKGROUND-COLOR: #ffffcc" type=button value="See It!">

<INPUT id=txtBusinessInfoName name=txtBusinessInfoName style="HEIGHT: 20px; LEFT: 123px; POSITION:
absolute; TOP: 400px; WIDTH: 250px; Z-INDEX: 103" value=TextField1>
<INPUT id=txtBusinessInfoAddress name=txtBusinessInfoAddress style="HEIGHT: 20px; LEFT: 123px;
POSITION: absolute; TOP: 420px; WIDTH: 250px; Z-INDEX: 103" value=TextField1>
<INPUT id=txtBusinessInfoPhone name=txtBusinessInfoPhone style="HEIGHT: 20px; LEFT: 123px; POSITION:
absolute; TOP: 440px; WIDTH: 250px; Z-INDEX: 103" value=TextField1>

<INPUT id=btnSwitchView name=btnSwitchView style="HEIGHT: 26px; LEFT: 507px; POSITION: absolute;
TOP: 342px; WIDTH: 74px; Z-INDEX:  104; BACKGROUND-COLOR: #ffffcc" type=button value="switch view" >
<INPUT id=btnMoveWest name=btnMoveWest style="HEIGHT: 24px; LEFT: 468px; POSITION: absolute; TOP:
343px; WIDTH: 31px; Z-INDEX: 105; BACKGROUND-COLOR: #ffffcc; FONT-WEIGHT: bold" type=button
value="W">
<INPUT id=btnMoveEast name=btnMoveEast style="HEIGHT: 24px; LEFT: 588px; POSITION: absolute; TOP:
343px; WIDTH: 34px; Z-INDEX:  106; BACKGROUND-COLOR: #ffffcc; FONT-WEIGHT: bold" type=button
value="E">
<INPUT id=btnMoveNorth name=btnMoveNorth style="HEIGHT: 24px; LEFT: 526px; POSITION: absolute; TOP:
313px; WIDTH: 35px; Z-INDEX: 107; BACKGROUND-COLOR: #ffffcc; FONT-WEIGHT: bold" type=button value=N>
<INPUT id=btnMoveSouth name=btnMoveSouth style="HEIGHT: 24px; LEFT: 526px; POSITION: absolute; TOP:
372px; WIDTH: 35px; Z-INDEX: 108; BACKGROUND-COLOR: #ffffcc; FONT-WEIGHT: bold" type =button value="
S ">


<IMG id=imgVederiLogo name=imgVederiLogo src="" sytle="LEFT: 5px; TOP: 0px; POSITION: absolute">

<IMG id=imgCarIcon name=imgCarIcon src="" style="LEFT: 285px; POSITION: absolute; TOP: 404px; Z-
INDEX: 113">
<IMG id=imgDotIcon name=imgDotIcon src="" style="LEFT: 338px; POSITION: absolute; TOP: 403px; Z-
INDEX: 114">



<OBJECT classid=CLSID:48E59293-9880-11CF-9754-00AA00C00908 id=Inet1
style="LEFT: 0px; TOP: 0px">
</OBJECT>

<TITLE>Vederi.com ScoutTool.  See where you're going!</TITLE>

</BODY></HTML>
```

Appendix: code for creating GPS coordinate spline functions

```
% GenerateSplines.m
% LG000903 : fix inaccuracy in t_given_p splines; cubic splines do a bad job of
interpolation
%            when we stand  still; function becomes discontinuous there.
%            Solution: use a smoothing spline.  Turns out hard to get a cubic smoothing
spline to
%                      work well due to the discontinuity when car is stopped, so use a
linear
%                      smoothing spline.

% LG000802 (originaly called Example_computeTofS.m)
% NOTE: I have modified this file so that it will
%       run through all the sequences and compute the splines,
%       and save them to file.
%
%       The filenames are of format :  seqNNTgivenP and the variable saved is
TgivenPspline
%                                      seqNNPgivenT, variable PgivenTspline,  where NN is
1 to 17
%       (TgivenP means 'Time given Position'; the video time corresponding to a given
%        (absolute) position measured along arc length of camera path)
%
%       These splines are used by \panorama\synthPan to generate synthetic perspective
panoramic
%       views.

%% Constants
ORIGIN_X = -2.06167581109917; %% Used as origin for the mercator coordinates
ORIGIN_Y = 0.63502761200326;  %% Computed as the mean of the coordinates in seq.
June27GPS1
MIN_SAMPLES_DIST = 1e-7; %% Minimum distance between two GPS adjacent samples
                            %% in the mercator coordinate system. Used to eliminate
                          %% samples generated while the car was stationary
SCALE = 5272498.48904543;  %% m/mercator

%%% List of files to be processed
filename{1} = 'June27GPS1';
filename{2} = 'June27GPS2';
filename{3} = 'June27GPS3';
filename{4} = 'June27GPS4';
filename{5} = 'June29GPS1';
filename{6} = 'June29GPS2';
filename{7} = 'June29GPS3';
filename{8} = 'June29GPS4';
filename{9} = 'June29GPS5';
filename{10} = 'July1GPS1';
filename{11} = 'July1GPS2';
filename{12} = 'July1GPS3';
filename{13} = 'July1GPS5';
filename{14} = 'July1GPS6';
filename{15} = 'July3GPS1';
filename{16} = 'July3GPS2';
filename{17} = 'July3GPS3';

%% Set the axis to convert the input coordinates from geographic to mercator
figure(1);clf;
axesm('MapProjection','mercator');

%%% Load the calibration file
load c:\users\luis\matlab\fromEnrico\svFileListCalib.txt

%%% CHOOSE file to generate spline for
```

```
for fileind = 1:17
    fprintf('NOTE: NOT computing all splines!!!\n');

eval(['fh = fopen(''e:\GPScoords\',filename{fileind},'.txt'',''rt'');']);

nsamples = fscanf(fh,'%d',1);
coord = reshape(fscanf(fh,'%f',3*nsamples),3,nsamples)';

%%% The time of the first sample should be read before calling the cleanUpGPSData
function because this is
%%% the sample that Luis used to compute the calibration
FirstSampleUTCTime = coord(1,1);

%%% Convert to mercator coordinates and subtract the mean
figure(1);clf;
axesm('MapProjection','mercator');
[coordXp,coordYp] = mfwdtran(coord(:,2),coord(:,3));
coordXp = coordXp - ORIGIN_X;
coordYp = coordYp - ORIGIN_Y;
coordTp = coord(:,1);
%keyboard

%%% Clean up data: replace clusters of samples with just the first point of the cluster
% LG000903 : No, cannot remove datapoints;  or we loose information (won't know exactly
when
%             we started moving)
%disp('...cleaning up samples');
%[coordX,coordY,Samples_Ind] = cleanUpGPSData(coordX, coordY, MIN_SAMPLES_DIST);
%coordT = coord(Samples_Ind,1);
[coordT,coordX,coordY] = adjustRepeatedGPSdatapoints(coordTp,coordXp,coordYp);

%%% Convert from UTC time to video time using the calibration data
coordT_V = coordT - FirstSampleUTCTime + svFileListCalib(fileind);

%%% Compute the spline y = f(x) where x is arc length in meters and y is Video time in
seconds
s_given_t = computeSofT2(coordT_V, coordX, coordY, SCALE);

PgivenTspline = s_given_t;

%fprintf('NOTE: NOT saving p_given_t data!!!\n');

eval(['save seq' num2str(fileind) 'PgivenT PgivenTspline']);

% LG: Compute spline for time (in seconds)  given arc length position (in meters)
% LG000903: use smoothing 1st order spline, with 0.1s tolerance for fit
TgivenPspline = computeTgivenPsmoothing(coordT_V, coordX, coordY, SCALE, 0.1);

eval(['save seq' num2str(fileind) 'TgivenP TgivenPspline']);

%keyboard


end;
```

```
function [coordT,coordX,coordY] = adjustRepeatedGPSdatapoints(coordTp,coordXp,coordYp)
% LC000903
% Need to get rid of datapoints that have the same exact GPS coordinates because it
screws
% up the time_given_position spline (we don't have a one-to-one function anymore).
% Exact same GPS coordinates can happen when we are completely stopped.
%
%Algorithm: rather than remove points (which is not good, since we need to know at what
times
% the car is stopped), adjust the GPS coordinates with an infinitessimal increment

%first delete any datapoints with the exact same time! (ie, repeat transmit/acquisition
of GPS data)
delT = [99; diff(coordTp)];
indGood = find(delT~=0);
coordTg = coordTp(indGood);
coordXg = coordXp(indGood);
coordYg = coordYp(indGood);

%find datapoints with same exact coords
% add a dummy entry at beginning so that vectors are same size, and the first repeat is
signaled
% with diff = 0
delX = [99; diff(coordXg)];
delY = [99; diff(coordYg)];

same = (delX==0).*(delY==0);

shift = cumsum(same);

% we will add a shift that is one millionth the size of the smallest non-zero distance
between datapoints
smallX = max([min(abs(delX(find(delX~=0))))/1e6 eps]);
smallY = max([min(abs(delY(find(delY~=0))))/1e6 eps]);

coordX = coordXg + smallX*shift;
coordY = coordYg + smallY*shift;
coordT = coordTg;

keyboard;
```

```
function sfunt_spline = computeTgivenPsmoothing(coordT, coordX, coordY, SCALE,
tolerance);
% LG000903 try a smoothing rather than interpolating spline.
% LG000802 modified from Enrico's computeSofT2.m
% tolerance input parameter specifies amount of smoothing (tolerance is level of error in
meters)

%%% Given a set of [X Y t] coordinates computes the t = f(s) time spline function where s
is arch length in meters
%%% and t is time in seconds

ds = sqrt(sum(diff([coordX coordY],1,1).^2,2));

ds = [ds(1); ds];

s = cumsum(ds)*SCALE;

sfunt_spline = spaps(s, coordT, tolerance,1); % 1 - linear spline!


function sfunt_spline = computeSofT2(coordT, coordX, coordY, SCALE);

%%% Given a set of [X Y t] coordinates computes the t = f(s) time spline function where s
is arch length in meters
%%% and t is time in seconds

ds = sqrt(sum(diff([coordX coordY],1,1).^2,2));

ds = [ds(1); ds];

s = cumsum(ds)*SCALE;

sfunt_spline = csapi(coordT, s);
```

Appendix: code for computing synthetic panoramas

```
function [pixFrameVector,firstPixTime,firstPixFrame] = generatePixelVector(sequenceID,t0)
%LG000809 generatePixelVector.m (copied and modified from generatePixelTimes)
%LG000802 generatePixelTimes.m  (copied from synthPan.m) (see notes 000802.1)
%
% generatePixelVector:
% For use by Enrico's VB program that uses a DirectSnow filter graph to access
% .avi files efficiently and build the synthetic image.
%
% Input :  sequenceID : to identify the video and GPS sequence and access the appropriate
%                       pos(time) and time(pos) splines
%          t0 : time in video corresponding to the center of the panorama
%
% Output:  pixFrameVector(1:720) :  pixFrameVector(col) is the relative frame and field
from which
%                                to get the pixel for column 'col' of the synthetic
panorama.
%                                The first frame is frame 1. (0 and -0 wouldn't work, as
Enrico pointed out!)
%                                If the value is Positive,
%                                then the Even field (2nd, 4th, 6th row of image) should be
used.
%                                If the value is Negative,
%                                Then the Odd field (1st, 3rd, 5th row of image) should be
used.
%
%          firstPixTime : the time (seconds) in the .avi of the first frame to be used
%                         (the .avi frame that will be used to generate the 1st column of
the panorama)
%
% NOTE: assumptions:
% Panoramic image is 720x240
% distance covered along scan path by panorama: 48m
% Frame rate of video : 30 fps ???is it really, I've seen 29.97 somewhere???
%
% generatePixelTimes:
% For use by Visual Basic.
% Algorithm:
%    For each pixel(column) in the synthetic view, compute a corresponding video time.
%    This is done using the GPS data and Enrico's routines that generate a spline
function
%    which computes time given arc length position along path.
%    Then we can either use optical flow to compute pixel data with subframe accuracy, or
%    round off each column's time to the nearest frame and copy that frame's column data.

%%% TESTING:
% sequenceID = 1; % June27DV1
% t0 = 3700/30; % center on frame 3700

% Input:
%     sequenceID : index specifying which video sequence to use
%
%     t0         : center time (seconds).
%                  Time in video corresponding to the center column of image.

% Output:
%     t(pixel)   : t(1:720), the time (seconds) corresponding to each column of the
synthetic view.
%     synth      : the synthetic image (done in Visual Basic)

% Variables
%
% f  (pixels)  : focal length of camera used
% F  (meters)  : focal distance of new synthetic view
```

```
%                 (ie, perpendicular distance from new focal point to trajectory of video
sequence)
% d(pixel)      : [meters] the arc length distance of each pixel column from the center
time position
% d0            : [meters] the absolute arc length corresponding to center time t0

% Some Constants
% NOTE: frame rate is 29.970, not 30.000 !!!
FRAME_RATE = 29.970; % [frames per second]
CAMERA_WIDTH = 720; % number of pixels in an image
CAMERA_HEIGHT = 480;
CAMERA_FOCAL = 48; % [mm], equivalent for 35 mm film (from SONY TRV-120 manual)

desiredStreetView = 48; % [meters] amount of street that must be visible in synthetic
view


% Choose a distance for the synthetic view
% If we need to see 48 meters of street to guarantee that the house is in view,
% then the closest distance for the synthetic view is
% NOTE: could subtract out distance of camera to house fronts, i.e., subtract 5 meters
F = CAMERA_FOCAL/35*48;

% Compute focal length of camera in pixels
% from manual: focal length in wide is equiv of 48mm for 35mm film -> FOV is
40.06202536179546 deg
% halfWidth/f = (35/2)/48 so
f = CAMERA_FOCAL/(35)*(CAMERA_WIDTH);

% Compute arc length distance from center time for each column of the synthetic image
% If pixels go from 1 to 720, center pixel is 360.5
pixels = [1:CAMERA_WIDTH] - (1+CAMERA_WIDTH)/2;
d = pixels/f*F;

%%%%% Compute corresponding video time for each pixel column of the synthetic image

% First load in appropriate time_given_position spline function
%TgivenPspline = loadTgivenPspline(sequenceID);
eval(['load c:\users\luis\matlab\fromEnrico\seq' num2str(sequenceID) 'TgivenP']);
%PgivenTspline = loadPgivenTspline(sequenceID);
eval(['load c:\users\luis\matlab\fromEnrico\seq' num2str(sequenceID) 'PgivenT']);

% Next compute absolute path position corresponding to desired center time t0
d0 = fnval(PgivenTspline,t0);
% and compute absolute path positions for the synthetic image columns
d_abs = d + d0;

% Finally, compute the corresponding video times
t = fnval(TgivenPspline,d_abs)';

% Convert times to frames
frames = t*FRAME_RATE;

roundedHalfFrames = round(frames*2)/2;

keyboard;

% LG00081C : due to an problem/inaccuracy in the spline generation, sometimes
% we move back in time. This is not good (especially for the graph filter which only
% moves forward in time).
% HACK solution: make roundedHalfFrames non-decreasing.
for i=2:length(roundedHalfFrames)
   if roundedHalfFrames(i)<roundedHalfFrames(i-1)
      roundedHalfFrames(i) = roundedHalfFrames(i-1);
   end;
end;
```

```
floorFrames = floor(roundedHalfFrames);

firstPixFrame = floorFrames(1);

firstPixTime = firstPixFrame/FRAME_RATE;

roundedHalfFramesRel = roundedHalfFrames-firstPixFrame + 1; % start counting with 1 and -
1

floorFramesRel = floor(roundedHalfFramesRel);

field = sign(rem(roundedHalfFramesRel,1)-0.25); % -1 odd, 1 even

pixFrameVector = floorFramesRel.*field;

return;
```

Appendix - Matlab code for Database creation

Filename: FindTurns.m

```
%%% Post-Process the GPS data
%%% Computes, for each sequence, a spline interpolation of the data and
the coordinates of the turns
%%% Output (to a file)
%%%     coord [3 x NSamples] decimal latitude, longitude and time in
seconds for each sample
%%%     traj_sp cubic spline which passes through the samples
%%%     curvC [1 x NumCoord] vector of curvilinear coordinates evenly
distributed along the spline
%%%     TurnInds pointers to the curvC vector of locations of turns in
the sequence

%%% CONSTANTS
MIN_SAMPLES_DIST = 1e-7; %% Minimum distance between two GPS adjacent
samples
                         %% in the mercator coordinate system. Used
to eliminate
                         %% samples generated while the car was
stationary
WIN_SIZE = 15;          %% Size of observation window used to compute
Cornericity
MIN_CORN = 1.5e-3;      %% Minimum cornericity required to classify a
local maxima in
                         %% in cornericity as a turn
MIN_BLOCK_SIZE = 1.0e-5; %% In the coordinates provided by the mercator
projection
                         %% Used to find the turns that belong to the
same intersection

ORIGIN_X = -2.06167581109917; %% Used as origin for the mercator
coordinates
ORIGIN_Y = 0.63502761200326;  %% Computed as the mean of the
coordinates in seq. June27GPS1

COLOR_SEQ = ['r' 'g' 'b' 'y' 'm' 'k'];

%%% List of files to be processed
filename{1} = 'June27GPS1';
filename{2} = 'June27GPS2';
filename{3} = 'June27GPS3';
filename{4} = 'June27GPS4';
filename{5} = 'June29GPS1';
filename{6} = 'June29GPS2';
filename{7} = 'June29GPS3';
filename{8} = 'June29GPS4';
filename{9} = 'June29GPS5';
filename{10} = 'July1GPS1';
filename{11} = 'July1GPS2';
filename{12} = 'July1GPS3';
filename{13} = 'July1GPS5';
filename{14} = 'July1GPS6';
filename{15} = 'July3GPS1';
```

```
filename{16} = 'July3GPS2';
filename{17} = 'July3GPS3';


for fileind = 1:length(filename)

%%% 1. Segment sequences to detect corners

%coord =
readGPSData('H:\users\dibe\GPSData\GPSData\','June27GPS1.dat');
%seqname = 'July3GPS1.txt';
seqname = filename{fileind};
cd h:\users\dibe\GPSData\matlab

if (exist([seqname,'Turns.mat'])==2)
   disp(['...sequence ',seqname,' has already been process.
Skipping']);
else
   cd h:\users\dibe\matlab\gps
   disp(['...loading sequence ',seqname]);
   eval(['fh =
fopen(''H:\users\dibe\GPSData\LookupTables\',seqname,'.txt'',''rt'');']
);

   nsamples = fscanf(fh,'%d',1);
   coord = reshape(fscanf(fh,'%f',3*nsamples),3,nsamples)';

   %%% Convert to mercator coordinates and subtract the mean
   figure(1);clf;
   axesm('MapProjection','mercator');
   [coordX,coordY] = mfwdtran(coord(:,2),coord(:,3));
   coordX = coordX - ORIGIN_X;
   coordY = coordY - ORIGIN_Y;

   %%% Last part of June29GPS4 is not good after sample 1747
   if (strcmp(seqname, 'June29GPS4'))
      coordX = coordX(1:1747);
      coordY = coordY(1:1747);
      coord = coord(1:1747,:);
   end;

   %%% Eliminating second half of seq July1DV6 because video sequence
is not good and was redone in July3DV3
   if (strcmp(seqname, 'July1GPS6'))
      coordX = coordX(1:318);
      coordY = coordY(1:318);
      coord = coord(1:318,:);
   end;

   %%% Clean up data: replace clusters of samples with just the first
point of the cluster
   disp('...cleaning up samples');
   [coordX,coordY] = cleanUpGPSData(coordX, coordY, MIN_SAMPLES_DIST);

   %%% Compute pp-spline
   traj_sp = cscvn([coordX coordY]');
   %traj_sp = csaps(coord(:,1)',[coordX coordY]',1);
```

```
%% Parameterize with arc length and define coordinate samples
maxINT = fnbrk(traj_sp,'interval');
numSamples = floor(10000/2.97*maxINT(2));
curvC = [0:maxINT(2)/numSamples:maxINT(2)];
newCoord = fnval(traj_sp,curvC);

figure(2);clf;
plot(coordX(:,1),coordY(:,1),'r+');
hold on;
%fnplt(traj_sp,'r');
plot(newCoord(1,:),newCoord(2,:));
drawnow;
axis('equal');
zoom on;

%%% Compute pp-spline derivative
dCds_sp = fnder(traj_sp);

%% Compute Cornericity
disp('...finding turns');
Cornericity = zeros(1,length(curvC));
for ind = (WIN_SIZE+1)/2:length(curvC)-(WIN_SIZE+1)/2
%   disp(ind);
   [U,S,V] = svd(fnval(dCds_sp,curvC(ind-(WIN_SIZE-
1)/2:ind+(WIN_SIZE-1)/2))');
   Cornericity(ind) = S(2,2);
end;

%% Find corners as local maxima of Cornericity
l_Corn = [Cornericity(2:end) Cornericity(end)];
r_Corn = [Cornericity(1) Cornericity(1:end-1)];
CornerInd = find((Cornericity > l_Corn)&(Cornericity >
r_Corn)&(Cornericity > MIN_CORN));
CornerVal = fnval(traj_sp,curvC(CornerInd));

figure(2);
for ind = 1:length(CornerInd)
  turn_h(ind) =
plot(CornerVal(1,ind),CornerVal(2,ind),'k+','MarkerSize',10,'LineWidth'
,3);
end;

bad_ind = [];
answ = input('Is there any turn you would like to eliminate ? (Y/N)
','s');
while ((answ == 'y')|(answ == 'Y'))
   disp('Click on the turn you want to eliminate and then hit
return');
   pause;
   ind = find(gco == turn_h);
    if (isempty(ind))
     disp('No turn was selected');
   else

plot(CornerVal(1,ind),CornerVal(2,ind),'m+','MarkerSize',13,'LineWidth'
,5);
```

```
        bad_ind = [bad_ind ind];
       end;

       answ = input('Is there any other turn you would like to eliminate
? (Y/N) ','s');
   end;

   disp(['Eliminating turns: ',num2str(bad_ind)]);

   TurnInds = CornerInd(setdiff([1:length(CornerInd)],bad_ind));

   disp(['Saving data for this sequence in
h:\users\dibe\GPSData\matlab\',seqname,'Turns']);
   eval(['save h:\users\dibe\GPSData\matlab\',seqname,'Turns coord
TurnInds traj_sp curvC;']);

end; %% end if exist ...
cd h:\users\dibe\matlab\GPS
end; % for fileind
```

Filename: FindIntersections.m

```
%% FindIntersections builds a big matrix of mutual distances between
Turns and groups them in intersections
%%
%%% Outputs
%%%     AllTurns
%%%           .Val [2x NumTurns] (Mercator coords - ORIGIN) of all the
turns in Pasadena
%%%           .Der [2x NumTurns] dX/dc dY/dc  for all the turns in
Pasadena
%%%           .seq [1x NumTurns] index of the original sequence each
turn belongs to
%%%     Intersection [1x NumIntersections]
%%%           .Turns [1 x numturns] indexes to the Turns variable of the
corners that belong to this intersection
%%%           .coord [2 x 1] (Mercator coords - ORIGIN) of the
intersection
%%%


%%% CONSTANTS
MIN_SAMPLES_DIST = 1e-7; %% Minimum distance between two GPS adjacent
samples
                         %% in the mercator coordinate system. Used
to eliminate
                         %% samples generated while the car was
stationary
WIN_SIZE = 15;          %% Size of observation window used to compute
Cornericity
MIN_CORN = 1.5e-3;      %% Minimum cornericity required to classify a
local maxima in
                         %% in cornericity as a turn
MIN_BLOCK_SIZE = 0.75e-5; %% In the coordinates provided by the
mercator projection
                         %% Used to find the turns that belong to the
same intersection
MIN_TURN_DIST = 1e-6;  %% Minimum distance between to turns of the same
intersection but different sides
ORIGIN_X = -2.06167581109917; %% Used as origin for the mercator
coordinates
ORIGIN_Y = 0.63502761200326;  %% Computed as the mean of the
coordinates in seq. June27GPS1

COLOR_SEQ = ['r' 'g' 'b' 'y' 'm' 'k'];

%%% List of files to be processed
filename{1} = 'June27GPS1';
filename{2} = 'June27GPS2';
filename{3} = 'June27GPS3';
filename{4} = 'June27GPS4';
filename{5} = 'June29GPS1';
filename{6} = 'June29GPS2';
filename{7} = 'June29GPS3';
filename{8} = 'June29GPS4';
filename{9} = 'June29GPS5';
filename{10} = 'July1GPS1';
```

```
filename{11} = 'July1GPS2';
filename{12} = 'July1GPS3';
filename{13} = 'July1GPS5';
filename{14} = 'July1GPS6';
filename{15} = 'July3GPS1';
filename{16} = 'July3GPS2';
filename{17} = 'July3GPS3';

figure(1);clf;
figure(2);clf;
Turns.Val = [];
Turns.Der = [];
Turns.seq = [];
for fileind = 1:length(filename)

   seqname = filename{fileind};

   disp(['...loading data for seq. ',seqname]);
   eval(['load h:\users\dibe\GPSData\matlab\',seqname,'Turns coord
TurnInds traj_sp curvC;']);

   %%% Convert to mercator coordinates and subtract the mean
   figure(1);clf;
   axesm('MapProjection','mercator');
   [coordX,coordY] = mfwdtran(coord(:,2),coord(:,3));
   coordX = coordX - ORIGIN_X;
   coordY = coordY - ORIGIN_Y;

   newCoord = fnval(traj_sp,curvC);

   figure(2);
   plot(coordX(:,1),coordY(:,1),'r+');
   hold on;
   %fnplt(traj_sp,'r');
   plot(newCoord(1,:),newCoord(2,:));
   drawnow;
   axis('equal');
   zoom on;

   %%% Compute pp-spline derivative
   dCds_sp = fnder(traj_sp);

   thisTurnVal = fnval(traj_sp,curvC(TurnInds));
   thisTurnDer = fnval(dCds_sp,curvC(TurnInds));

   figure(2);
   plot(thisTurnVal(1,:),thisTurnVal(2,:),'k+','MarkerSize',10,'LineWid
th',3);

   Turns.Val = [Turns.Val thisTurnVal];
   Turns.Der = [Turns.Der thisTurnDer];
   Turns.seq = [Turns.seq fileind*ones(1,size(thisTurnVal,2))];

end;


%% Find intersections
```

```
disp('... building the big matrix of mutual distances');
numTurns = length(Turns.seq);
TurnDistMat = zeros(numTurns,numTurns);
for i = 1:numTurns
   for j = i:numTurns
      TurnDistMat(i,j) = norm(Turns.Val(:,i) - Turns.Val(:,j));
   end;
end;
tmpMat = TurnDistMat';
TurnDistMat = TurnDistMat + tril(tmpMat);
disp('...thresholding the matrix to group turns into intersections');
TurnCorrMat = (TurnDistMat~=0)&(TurnDistMat < MIN_BLOCK_SIZE);

disp('...scanning the correspondences to find unique intersections');

int_ind = 1;
AlreadyPicked = [];
Intersection = [];
for ind = 1:numTurns
   if ~ismember(ind,AlreadyPicked);
      corr_ind = FindCorrInd(ind,ind,TurnCorrMat);
      Intersection(int_ind).Turns = corr_ind;
      AlreadyPicked = union(AlreadyPicked, [ind corr_ind]);
      int_ind = int_ind + 1;
   end;
end;



disp('...computing the center of the intersection');
for ind = 1:length(Intersection)
   %Intersection(ind).coord =
mean(Turns.Val(:,Intersection(ind).Turns),2);
   turn_inds = Intersection(ind).Turns;
   turn_coords = Turns.Val(:,Intersection(ind).Turns);
   turn_slope =
Turns.Der(2,Intersection(ind).Turns)./Turns.Der(1,Intersection(ind).Tur
ns);

   turn_posslope_ind = turn_inds(find(turn_slope >= 0));
   turn_negslope_ind = turn_inds(find(turn_slope < 0));

   %%% Cluster turn_posslope_ind turns
   turn_dist_mat = TurnDistMat(turn_posslope_ind,turn_posslope_ind);
   turn_corr_mat = (turn_dist_mat~=0)&(turn_dist_mat < MIN_TURN_DIST);
   int_ind = 1;
   AlreadyPicked = [];
   inter_corner = [];
   for i = 1:length(turn_posslope_ind)
      if ~ismember(i,AlreadyPicked);
         corr_ind = FindCorrInd(i,i,turn_corr_mat);
         inter_corner(int_ind).Turns = turn_posslope_ind(corr_ind);
         AlreadyPicked = union(AlreadyPicked, [i corr_ind]);
         int_ind = int_ind + 1;
      end;
   end;
```

```
    %% Cluster turn_negslope_ind turns
    turn_dist_mat = TurnDistMat(turn_negslope_ind,turn_negslope_ind);
    turn_corr_mat = (turn_dist_mat~=0)&(turn_dist_mat < 1e-6);
    AlreadyPicked = [];
    for i = 1:length(turn_negslope_ind)
       if ~ismember(i,AlreadyPicked);
          corr_ind = FindCorrInd(i,i,turn_corr_mat);
          inter_corner(int_ind).Turns = turn_negslope_ind(corr_ind);
          AlreadyPicked = union(AlreadyPicked, [i corr_ind]);
          int_ind = int_ind + 1;
       end;
    end;

    %%% Average of clusters
    intersection_coords = [0;0];
    for i = 1:length(inter_corner)
       intersection_coords = intersection_coords +
mean(Turns.Val(:,inter_corner(i).Turns),2);
    end;
    Intersection(ind).coord = intersection_coords/length(inter_corner);

end;

%% Manually add two intersections through which I always drove streight
Intersection(end+1).coord = 1e-3*[-0.1490;
                                  -0.3657];
Intersection(end).Turns = [];

Intersection(end+1).coord = 1e-3*[-0.1491;
                                  -0.4109];
Intersection(end).Turns = [];

disp('...plotting the result');
figure(2)
tmp_coord = cat(2,Intersection.coord);
plot(tmp_coord(1,:),tmp_coord(2,:),'g+','MarkerSize',10,'LineWidth',5);
for ind = 1:length(Intersection)
   text(tmp_coord(1,ind),tmp_coord(2,ind),num2str(ind));
end;

disp('...saving the result');
AllTurns = Turns;

save h:\users\dibe\GPSData\matlab\PasadenaIntersections Intersection
AllTurns
```

Filename: FindESRIIntersections.m

```
%function [Intersections] = FindESRIIntersections()

%%% Parse through the ESRI segments list and find vertices with more
than 2 arcs

%% Output
%%   ESRIIntersection [NumIntersections x 1]
%%         .Coord = [lat long] in NAD83 datum
%%         .Segments = list of indexes to the Street segments that
converge to the intersection
%%         .NumSegments = length(.Segments)


%load h:\users\dibe\GPSData\matlab\PasadenaStreets Street;
load h:\users\dibe\GPSData\matlab\ESRIstreetsPasadena;
Street = streets;
load h:\users\dibe\GPSData\matlab\ESRIstreetsSanMarino;
Street = [Street streets];

NumSegments = length(Street);
%SegmentStarts = cat(1,Street.StartCoord);
SegmentStarts = [cat(1,Street.startLatitude)
cat(1,Street.startLongitude)];
%SegmentStops = cat(1,Street.StopCoord);
SegmentStops = [cat(1,Street.stopLatitude)
cat(1,Street.stopLongitude)];

Vertices = [SegmentStarts;SegmentStops];

figure(3);
plot([SegmentStarts(:,1) SegmentStops(:,1)]',[SegmentStarts(:,2)
SegmentStops(:,2)]')

AVCoord = sum(Vertices,1)/(2*NumSegments);
dist = sqrt(sum((Vertices - ones(2*NumSegments,1)*AVCoord).^2,2));

[tmp, S_ind] = sort(dist);
OVertices = Vertices(S_ind,:);

OVertices_d = [0 0; OVertices(1:end-1,:)];

diffVertices = sqrt(sum((OVertices - OVertices_d).^2,2));

diffVertices_1 = [diffVertices(2:end,1) ; 1];

StartInd = find((diffVertices ~= 0) & (diffVertices_1 == 0));
StopInd = find((diffVertices == 0) & (diffVertices_1 ~= 0));

NumVertices = StopInd - StartInd + 1;

IntersInd = find(NumVertices > 2);

for ind = 1:length(IntersInd)
   Intersection(ind).Coord = OVertices(StartInd(IntersInd(ind)),:);
   tmpind = S_ind(StartInd(IntersInd(ind)):StopInd(IntersInd(ind)));
   tmpind = tmpind - NumSegments*(tmpind > NumSegments);
```

```
   Intersection(ind).Segments = tmpind;
   Intersection(ind).NumSegments = NumVertices(IntersInd(ind));
end;


ESRIIntersection = Intersection;

ESRIIntersectionCoord = cat(1,ESRIIntersection.Coord);
figure(3);hold on;
plot(ESRIIntersectionCoord(:,1),ESRIIntersectionCoord(:,2),'bs','Marker
Size',5);

disp(['ESRIIntersection saved in PasadenaESRIIntersection file']);
```

Filename: FindSegments

```
%%% CONSTANTS
MIN_SAMPLES_DIST = 1e-7; %% Minimum distance between two GPS adjacent
samples
                         %% in the mercator coordinate system. Used
to eliminate
                         %% samples generated while the car was
stationary
WIN_SIZE = 15;          %% Size of observation window used to compute
Cornericity
MIN_CORN = 1.5e-3;      %% Minimum cornericity required to classify a
local maxima in
                         %% in cornericity as a turn
MIN_BLOCK_SIZE = 0.4e-5; %% In tne coordinates provided by the mercator
projection
                         %% Used to find the turns that belong to the
same intersection

ORIGIN_X = -2.06167581109917; %% Used as origin for the mercator
coordinates
ORIGIN_Y = 0.63502761200326;  %% Computed as the mean of the
coordinates in seq. June27GPS1

COLOR_SEQ = ['r' 'g' 'y' 'm' 'k'];


%%% List of files to be processed
filename{1} = 'June27GPS1';
filename{2} = 'June27GPS2';
filename{3} = 'June27GPS3';
filename{4} = 'June27GPS4';
filename{5} = 'June29GPS1';
filename{6} = 'June29GPS2';
filename{7} = 'June29GPS3';
filename{8} = 'June29GPS4';
filename{9} = 'June29GPS5';
filename{10} = 'July1GPS1';
filename{11} = 'July1GPS2';
filename{12} = 'July1GPS3';
filename{13} = 'July1GPS5';
filename{14} = 'July1GPS6';
filename{15} = 'July3GPS1';
filename{16} = 'July3GPS2';
filename{17} = 'July3GPS3';

%%% Load intersections
load h:\users\dibe\GPSData\matlab\PasadenaIntersections Intersection
AllTurns

IntersectionCoords = cat(2,Intersection.coord);
Intersection(1).Segments = [];
figure(1);clf;
figure(2);clf;

segm_ind = 1;  % init the segment index
```

```
Turns = []; %%% Big array of turns where I accumulate the turns of all
sequences
for fileind = 1:length(filename)

   seqname = filename{fileind};
   disp(['...loading sequence ',seqname, ' Turns file']);
   %%% Load pre-processed data
   eval(['load h:\users\dibe\GPSData\matlab\',seqname,'Turns coord
TurnInds traj_sp curvC;']);

   %%% Convert to mercator coordinates and subtract the mean
   figure(1);clf;
   axesm('MapProjection','mercator');
   [coordX,coordY] = mfwdtran(coord(:,2),coord(:,3));
   coordX = coordX - ORIGIN_X;
   coordY = coordY - ORIGIN_Y;

   %%% Clean up data: replace clusters of samples with just the first
point of the cluster
   disp('...cleaning up samples');
   [coordX,coordY] = cleanUpGPSData(coordX, coordY, MIN_SAMPLES_DIST);

   %% Parameterize with arc length and define coordinate samples
   maxINT = fnbrk(traj_sp,'interval');
   numSamples = floor(10000/2.97*maxINT(2));
   curvC = [0:maxINT(2)/numSamples:maxINT(2)];
   newCoord = fnval(traj_sp,curvC);

   figure(2);
   plot(coordX(:,1),coordY(:,1),'r+');
   hold on;
   %fnplt(traj_sp,'r');
   plot(newCoord(1,:),newCoord(2,:));
   drawnow;
   axis('equal');
   zoom on;

   %%% Compute pp-spline derivative
   dCds_sp = fnder(traj_sp);

   %% Compute Cornericity
   Cornericity = zeros(1,length(curvC));
   for ind = (WIN_SIZE+1)/2:length(curvC)-(WIN_SIZE+1)/2
   %   disp(ind);
      [U,S,V] = svd(fnval(dCds_sp,curvC(ind-(WIN_SIZE-
1)/2:ind+(WIN_SIZE-1)/2))');
      Cornericity(ind) = S(2,2);
   end;

   %%% Find entry and exit indexes of turns
   %% Find entry and exit arc length coord for each turn
   disp('...finding entry and exit indexes of turns');
   clear Turn;
   for ind = 1:length(TurnInds)
      Thresh = 0.75*Cornericity(TurnInds(ind));
      Turn(ind).CornerInd = TurnInds(ind);
      tmpind = TurnInds(ind);
```

```
      while ((Cornericity(tmpind) > Thresh)&(tmpind > 0))
          tmpind = tmpind - 1;
      end;
      Turn(ind).EntryInd = tmpind;
      tmpind = TurnInds(ind);
      while ((Cornericity(tmpind) > Thresh)&(tmpind < length(curvC)))
         tmpind = tmpind + 1;
      end;
      Turn(ind).ExitInd = tmpind;
      Turn(ind).Sequence = fileind;
   end;


   %% Time to segment the sequence !!!!
   color_ind = 0;
   for turn_ind = 1:length(Turn)-1
      start_ind = Turn(turn_ind).ExitInd+1;
      stop_ind = Turn(turn_ind+1).EntryInd-1;
      if (stop_ind <= start_ind)
         disp('Start index of segment is smaller than Stop index.
Probably due to turn overlap.');
         %keyboard;
      else
         SegmentCoordVal = fnval(traj_sp,curvC(start_ind:stop_ind));
         Segment_Length = sqrt(sum(SegmentCoordVal(:,1)-
SegmentCoordVal(:,end)).^2);
         if (Segment_Length < MIN_BLOCK_SIZE)
           disp(['... this segment is only ',num2str(Segment_Length),'
long. Check what''s going on']);
         else
            clear dist
            %for int_ind = 1:length(Intersection)
            %  dist(int_ind,:) = sqrt(sum((SegmentCoordVal -
Intersection(int_ind).coord*ones(1,(stop_ind - start_ind + 1))).^2,1));
            %end;
            dist =
sqrt((ones(length(Intersection),1)*SegmentCoordVal(1,:) -
IntersectionCoords(1,:)'*ones(1,stop_ind - start_ind + 1)).^2 + ...

   (ones(length(Intersection),1)*SegmentCoordVal(2,:) -
IntersectionCoords(2,:)'*ones(1,stop_ind - start_ind + 1)).^2);


            NearIntersections = find(min(dist,[],2)<1.0e-5);
            if(length(NearIntersections) < 2)
                disp(['...mmmmh, something is wrong. Segment between
turn ',num2str(turn_ind),...
                ' and turn ',num2str(turn_ind+1),' has less than two
near intersections !']);
                %keyboard;
            elseif (length(NearIntersections) == 2)
               %% This is a single segment. I can store it in the
segments list
               disp(['Segment ',num2str(segm_ind),' detected']);
               Segments.Start(segm_ind) = start_ind;
               Segments.Stop(segm_ind) = stop_ind;
               %Segments.Coord(segm_ind,:) = SegmentCoordVal;
```

```
                [tmpval,Segments.FromIntersection(segm_ind)] =
min(dist(:,1));
                [tmpval,Segments.ToIntersection(segm_ind)] =
min(dist(:,end));
                Segments.FromTurn(segm_ind) = turn_ind +
min(find(AllTurns.seq == fileind)) - 1;
                                                %%% turn_ind is the
turn index only for the curr sequence
                Segments.ToTurn(segm_ind) = turn_ind + 1 +
min(find(AllTurns.seq == fileind)) - 1;
                Segments.Sequence(segm_ind) = fileind;

                %%% Adding the segment index in the Intersection list

Intersection(Segments.FromIntersection(segm_ind)).Segments =
union(Intersection(Segments.FromIntersection(segm_ind)).Segments,...
                                                                segm_ind);
                Intersection(Segments.ToIntersection(segm_ind)).Segments
= union(Intersection(Segments.ToIntersection(segm_ind)).Segments,...
                                                                segm_ind);
                %%% Adding the segment index in the Turn list
                Turn(turn_ind).ToSegment = segm_ind;
                Turn(turn_ind+1).FromSegment = segm_ind;

                %%% Adding the Intersection index in the Turn list
                Turn(turn_ind).Intersection =
Segments.FromIntersection(segm_ind);
                Turn(turn_ind).ToIntersection =
Segments.ToIntersection(segm_ind);
                Turn(turn_ind+1).Intersection =
Segments.ToIntersection(segm_ind);

                disp(['     From Intersection =
',num2str(Segments.FromIntersection(segm_ind)),...
                      'To Intersection =
',num2str(Segments.ToIntersection(segm_ind))]);
                segm_ind = segm_ind + 1;
                %% Plot the segment
                figure(2);

   plot(SegmentCoordVal(1,:),SegmentCoordVal(2,:),COLOR_SEQ(rem(color_i
nd,5)+1));
                color_ind = color_ind+1;
             elseif (length(NearIntersections) > 2)
                %% This segment need to be broken up in single block
segments
                disp(['Breaking up segment in
',num2str(length(NearIntersections)-1),' subsegments']);
                %%
                [tmpval,min_ind] = min(dist,[],2);
                [break_ind,sort_ind] = sort(min_ind(NearIntersections));

                for ind = 1:(length(NearIntersections)-1)
                   start_ind1 = start_ind + break_ind(ind) - 1;
                   stop_ind1 = start_ind + break_ind(ind+1) - 1;
                   Segments.Start(segm_ind) = start_ind1;
                   Segments.Stop(segm_ind) = stop_ind1;
```

```
                    SegmentCoord =
SegmentCoordVal(:,break_ind(ind):break_ind(ind+1));
                    Segments.FromIntersection(segm_ind) =
NearIntersections(sort_ind(ind));
                    Segments.ToIntersection(segm_ind) =
NearIntersections(sort_ind(ind+1));
                    disp(['      Segment ',num2str(segm_ind),'
detected']);
                    disp(['            Start Intersection =
',num2str(Segments.FromIntersection(segm_ind)),...
                          'Stop Intersection =
',num2str(Segments.ToIntersection(segm_ind))]);
                    if (ind == 1)
                       Segments.FromTurn(segm_ind) = turn_ind +
min(find(AllTurns.seq == fileind)) - 1;
                       %%% turn_ind is the turn index only for the curr
sequence
                    else
                       Segments.FromTurn(segm_ind) = 0;
                    end;
                    if (ind == (length(NearIntersections)-1))
                       Segments.ToTurn(segm_ind) = turn_ind + 1 +
min(find(AllTurns.seq == fileind)) - 1;
                    else
                       Segments.ToTurn(segm_ind) = 0;
                    end;
                    Segments.Sequence(segm_ind) = fileind;

                    %%% Adding the segment index in the Intersection list

   Intersection(Segments.FromIntersection(segm_ind)).Segments =
union(Intersection(Segments.FromIntersection(segm_ind)).Segments,...
                                                                    segm_ind);

   Intersection(Segments.ToIntersection(segm_ind)).Segments =
union(Intersection(Segments.ToIntersection(segm_ind)).Segments,...
                                                                    segm_ind);
                    %%% Adding the segment index in the Turn list and the
intersection index in the Turn list
                    if (ind == 1)
                       Turn(turn_ind).ToSegment = segm_ind;
                       Turn(turn_ind).Intersection =
Segments.FromIntersection(segm_ind);
                       Turn(turn_ind).ToIntersection =
Segments.ToIntersection(segm_ind);
                    elseif (ind == (length(NearIntersections)-1))
                       Turn(turn_ind+1).FromSegment = segm_ind;
                       Turn(turn_ind+1).Intersection =
Segments.ToIntersection(segm_ind);
                    end;

                    %% Plot the segment
                    figure(2);

   plot(SegmentCoord(1,:),SegmentCoord(2,:),COLOR_SEQ(rem(color_ind,5)+
1));
                    drawnow;
```

```
                    %% Increase tne segment index
                    color_ind = color_ind+1;
                    segm_ind = segm_ind + 1;
                end;
            end;
        end;
    end;

    if (isempty(Turns))
        Turns = Turn;
    else;
        Turns = [Turns Turn];
    end;

end;

save h:\users\dibe\gpsdata\matlab\Pasadena_Segments_Intersections_Turns
Segments Intersection Turns
```

Filename: LabelSegments.m

```
%% Label Segments

%% Goes through the list of Segments and for each segment
%%% 1. Finds the start and stop intersections
%%% 2. Finds the corresponding ESRI Intersections
%%% 3. Find the two ESRI feature elements that are contained in the
original segment
%%% 4. Use those segments to label the original segment, adding the
street name and the start and stop left/right
%%%    street numbers
%%% 5. Track the ESRI feature elements to find all the feature elements
that belong to the original segment
%%% 6. Add to the Street array the field Segment #
%%% 7. Add to the segment array the field FeatureElInds and
FeatureElOrient for the feature elements contained in that segment

%%% EDB071400 7PM: finally I managed to label most of the segment (71
segments could not be labeled because one of
%%%           the two intersection does not mat to an ESRI Intersection
(there are missing streets in the ESRI data)
%%%           I had to manually fix few wrong correspondences between
Intersections and ESRIIntersections.

%%% Load GPS derived data
load
h:\users\dibe\GPSData\matlab\Pasadena_Segments_Intersections_Turns2
Intersection Turns Segments

%%% Load ESRI data
   %% Load ESRI Intersections
   load h:\users\dibe\GPSdata\matlab\PasadenaESRIIntersection
ESRIIntersection

   %% Load ESRI data
   load h:\users\dibe\GPSData\matlab\ESRIstreetsPasadena;
   Street = streets;
   load h:\users\dibe\GPSData\matlab\ESRIstreetsSanMarino;
   Street = [Street streets];

NumSegments = length(Segments.Start);

ESRIFeatureElStartCoords = [cat(1,Street.startLatitude)
cat(1,Street.startLongitude)];
ESRIFeatureElStopCoords = [cat(1,Street.stopLatitude)
cat(1,Street.stopLongitude)];


%%% display data
DisplayGPSandESRI;
figure(2);
segmh = plot([0 0],[0 0],'g','LineWidth',4);
EInth = plot(0,0,'rs');
Inth = plot(0,0,'r*');


%% Init field Segment in Street array
```

```
Street(1).Segment = [];

numunlabeled = 0;
for segm_ind = 1:NumSegments
   %%% Copy in MySegment the already existing fields in Segments
   MySegment(segm_ind).Start = Segments.Start(segm_ind);
   MySegment(segm_ind).Stop = Segments.Stop(segm_ind);
   MySegment(segm_ind).FromIntersection =
Segments.FromIntersection(segm_ind);
   MySegment(segm_ind).ToIntersection =
Segments.ToIntersection(segm_ind);
   MySegment(segm_ind).FromTurn = Segments.FromTurn(segm_ind);
   MySegment(segm_ind).ToTurn = Segments.ToTurn(segm_ind);
   MySegment(segm_ind).Sequence = Segments.Sequence(segm_ind);

   %%% From and To Intersections
   FromInt = Segments.FromIntersection(segm_ind);
   ToInt = Segments.ToIntersection(segm_ind);
   %%% Corresponding ESRI Intersections
   EFromInt = Intersection(FromInt).ESRIIntersection;
   EToInt = Intersection(ToInt).ESRIIntersection;
   if (isempty(EFromInt)|isempty(EToInt)|(EFromInt(1) == 0)|(EToInt(1)
== 0))
      disp(['... I cannot label segment ',num2str(segm_ind),...
            ' because one of the intersections does not correspond to
an ESRI Int']);
      numunlabeled = numunlabeled + 1;
   else
      %%% ESRI Feature elements going to the ESRI Intersections
      FromFeatureElInd = ESRIIntersection(EFromInt(1)).Segments;
      if (EFromInt(2) ~= 0)  %% more than one ESRI intersection
corresponds to this intersection
         FromFeatureElInd = union(FromFeatureElInd ,
ESRIIntersection(EFromInt(2)).Segments);
      end;
      ToFeatureElInd = ESRIIntersection(EToInt(1)).Segments;
      if (EToInt(2) ~= 0)   %% more than one ESRI intersection
corresponds to this intersection
         ToFeatureElInd = union(ToFeatureElInd,
ESRIIntersection(EToInt(2)).Segments);
      end;

      %%% The name of the street is the common denominator of the two
groups of feature elements
      LabelMatch =
zeros(length(FromFeatureElInd),length(ToFeatureElInd));
      for i = 1:length(FromFeatureElInd)
          for j = 1:length(ToFeatureElInd)
            LabelMatch(i,j) =
strcmp(Street(FromFeatureElInd(i)).name,Street(ToFeatureElInd(j)).name)
;
         end;
      end;
      [fromind,toind] = find(LabelMatch);
      if (isempty(fromind))
         disp('...mmmh, weird !? I could not find two feature els with
the same street name');
```

```
        % Display the segment
        eval(['load
h:\users\dibe\GPSData\matlab\',filename{Segments.Sequence(segm_ind)},'T
urns traj_sp curvC;']);
        SegmentCoords =
fnval(traj_sp,curvC(Segments.Start(segm_ind):Segments.Stop(segm_ind)));

set(segmh,'XData',SCALE*SegmentCoords(1,:),'YData',SCALE*SegmentCoords(
2,:));
        set(Inth,'XData',SCALE*[Intersection(FromInt).coord(1);...

Intersection(ToInt).coord(1)],'YData',SCALE*[Intersection(FromInt).coor
d(2);...
              Intersection(ToInt).coord(2)]);
        EFromIntCoords = ESRIIntersection(EFromInt(1)).Coord;
        numint = 1;
        if (EFromInt(2) ~= 0)
           EFromIntCoords =
[EFromIntCoords;ESRIIntersection(EFromInt(2)).Coord];
           numint = 2;
        end;
        figure(1);
        [EFromIntCoords(:,1),EFromIntCoords(:,2)] =
mfwdtran(EFromIntCoords(:,1),EFromIntCoords(:,2));
        EFromIntCoords = EFromIntCoords - ones(numint,1)*[ORIGIN_X
ORIGIN_Y];

%set(EInth,'XData',EFromIntCoords(:,1),'YData',EFromIntCoords(:,2));

        EToIntCoords = ESRIIntersection(EToInt(1)).Coord;
        numint = 1;
        if (EToInt(2) ~= 0)
           EToIntCoords =
[EToIntCoords;ESRIIntersection(EToInt(2)).Coord];
           numint = 2;
        end;
        figure(1);
        [EToIntCoords(:,1),EToIntCoords(:,2)] =
mfwdtran(EToIntCoords(:,1),EToIntCoords(:,2));
        EToIntCoords = EToIntCoords - ones(numint,1)*[ORIGIN_X
ORIGIN_Y];

set(EInth,'XData',SCALE*[EFromIntCoords(:,1);EToIntCoords(:,1)],'YData'
,SCALE*[EFromIntCoords(:,2);EToIntCoords(:,2)]);

        drawnow;
        keyboard;
        numunlabeled = numunlabeled + 1;
     else
        disp(['... labeling Segment ',num2str(segm_ind),' :
',Street(FromFeatureElInd(fromind(1))).name]);
        MySegment(segm_ind).StreetName =
Street(FromFeatureElInd(fromind(1))).name;
        %keyboard;
        fromind = unique(fromind);
        toind = unique(toind);
```

```
        %% Among the FromFeatureElInd(fromind) feature els that match
the name, which one belongs to the segment ?
        figure(1);
        clear FStartCoords FStopCoords;
        [FStartCoords(:,1),FStartCoords(:,2)] =
mfwdtran(cat(1,Street(FromFeatureElInd(fromind)).startLatitude),...

cat(1,Street(FromFeatureElInd(fromind)).startLongitude));
        FStartCoords = FStartCoords -
ones(length(fromind),1)*[ORIGIN_X ORIGIN_Y];
        [FStopCoords(:,1),FStopCoords(:,2)] =
mfwdtran(cat(1,Street(FromFeatureElInd(fromind)).stopLatitude),...

cat(1,Street(FromFeatureElInd(fromind)).stopLongitude));
        FStopCoords = FStopCoords - ones(length(fromind),1)*[ORIGIN_X
ORIGIN_Y];
        % Compute the distance from the ToInt intersection of each
start and stop coords
        dist = sqrt(sum(([FStartCoords;FStopCoords]' -
Intersection(ToInt).coord * ones(1,2*length(fromind))).^2,1));
        [minval,minind] = min(dist);
        if (minind <= length(fromind))
           firstel.ind = FromFeatureElInd(fromind(minind));
           firstel.orient = -1; %% (side closer to the to intersection
is the start)
        else
           firstel.ind = FromFeatureElInd(fromind(minind-
length(fromind)));
           firstel.orient = 1; %% (side closer to the to intersection
is the stop)
        end;
        MySegment(segm_ind).FeatureElInds = firstel.ind;
        MySegment(segm_ind).FeatureElOrient = firstel.orient;
        if (firstel.orient == 1)  %% Feature el oriented according to
direction of travel
           MySegment(segm_ind).FirstAddress =
Street(firstel.ind).L_F_ADDR;
        else                      %% Feature el oriented opposite to
direction of travel
           MySegment(segm_ind).FirstAddress =
Street(firstel.ind).R_T_ADDR;
        end;
        Street(firstel.ind).Segment = [Street(firstel.ind).Segment
segm_ind];
        %%% Now track the feature elements until I reach one of the
ToFeatureElInd(toind) elements
        curr_el.ind = firstel.ind;
        curr_el.orient = firstel.orient;
        while (sum(ismember(curr_el.ind, ToFeatureElInd(toind))) == 0)
           if (curr_el.orient == 1)
              nextelcoords = [Street(curr_el.ind).stopLatitude
Street(curr_el.ind).stopLongitude];
           else
              nextelcoords = [Street(curr_el.ind).startLatitude
Street(curr_el.ind).startLongitude];
           end;
```

```
            cand_ind_start = find((ESRIFeatureElStartCoords(:,1) ==
nextelcoords(1))&...
               (ESRIFeatureElStartCoords(:,2) == nextelcoords(2)));
            cand_ind_stop = find((ESRIFeatureElStopCoords(:,1) ==
nextelcoords(1))&...
               (ESRIFeatureElStopCoords(:,2) == nextelcoords(2)));
            %% Only one valid element should come out of this
            next_el.ind = [];
            next_el.orient = [];
            for i = 1:length(cand_ind_start)
               if ((cand_ind_start(i) ~=
curr_el.ind)&(strcmp(Street(cand_ind_start(i)).name,MySegment(segm_ind)
.StreetName)))
                  next_el.ind = [next_el.ind cand_ind_start(i)];
                  next_el.orient = [next_el.orient 1];
               end;
            end;
            for i = 1:length(cand_ind_stop)
               if ((cand_ind_stop(i) ~=
curr_el.ind)&(strcmp(Street(cand_ind_stop(i)).name,MySegment(segm_ind).
StreetName)))
                  next_el.ind = [next_el.ind cand_ind_stop(i)];
                  next_el.orient = [next_el.orient -1];
               end;
            end;
            MySegment(segm_ind).FeatureElInds =
[MySegment(segm_ind).FeatureElInds next_el.ind(1)];
            MySegment(segm_ind).FeatureElOrient =
[MySegment(segm_ind).FeatureElOrient next_el.orient(1)];
            Street(next_el.ind(1)).Segment =
[Street(next_el.ind(1)).Segment segm_ind];
            curr_el.ind = next_el.ind;
            curr_el.orient = next_el.orient;
            if (length(next_el.ind) > 1)
               disp('...mmh. There are more than one candidate to be
next el. How is that possible ?');
               keyboard;
            end;

         end;
         if (curr_el.orient == 1)  %% Feature el oriented according to
direction of travel
            MySegment(segm_ind).LastAddress =
Street(curr_el.ind(1)).L_T_ADDR;
         else                          %% Feature el oriented opposite to
direction of travel
            MySegment(segm_ind).LastAddress =
Street(curr_el.ind(1)).R_F_ADDR;
         end;

         %if (length(MySegment(segm_ind).FeatureElInds) > 4)
         %   disp('...mmmh. There are more than 4 feature elements in
this segments. Take a look Dave');
         %   keyboard;
         %end;

      end;
```

```
   end;
end;

%% Save the results
Segment = MySegment;

%save h:\users\dibe\GPSData\matlab\PasadenaAllData Segment Street
Intersection ESRIIntersection Turns


return;
```

Filename: GenSegmArray.m

```
% Generate the database of street segments to be used by the scouting
tool
% The main datastructure in the database is a structure array called
SegmArray containing the following fields:
%
%  Name                 Type                 Size
% ------------------------------------------------------
% StreetName          String                 1x1
% FromAddress          Int                   1x2     [Even,Odd]
% ToAddress            Int                   1x2
% SideofHub           String                 1x1    One of the
following: 'N','S','W','E'
%
% FromMapCoordGeo     Double                 2x1  [lat,long]'
% FromMapCoordMerc    Double                 2x1  [x, y]'
% ToMapCoordGeo       Double                 2x1
% ToMapCoordMerc      Double                 2x1
%
% FromSegment          Int                   [4x1] In order: E,N,W,S
% ToSegment            Int                   [4x1]
%
% DistanceFromHub     Double                 [1x2]
% Length              Double                 [1x2]
% Offset              Double                 [1x2]
% OffsetFlag           Int                   [1x2] One of the
following: 0=Don't know if it's right, 1=Hard constrain,
%
2=Impossible to determine,  3=Result of optimization process
% NumPanoramas         Int                   [1x2]
% PanoramaCoord       Double                 [Nx2] Distance from the
origin of the segment in meters
% PanoramaTime        Double                 [Nx2] N =
max(NumPanoramas)
% NumFrames            Int                   [1x2]
% FrameTime           Double                 [Mx2] M = max(NumFrames)
% FrameSeq             Int                   [1x2] Sequence numbers
starts from 1

% SegmIndex            Int                   [1x2] Original two
segments in the Segment array

% Some constants
ORIGIN_X = -2.06167581109917; %% Used as origin for the mercator
coordinates
ORIGIN_Y = 0.63502761200326;  %% Computed as the mean of the
coordinates in seq. June27GPS1
MIN_BLOCK_SIZE = 1.0e-5; %% In the coordinates provided by the mercator
projection
MIN_SAMPLES_DIST = 1e-7; %% Minimum distance between two GPS adjacent
samples
                         %% in the mercator coordinate system. Used
to eliminate
                         %% samples generated while the car was
stationary
SCALE = 5272498.48904543;  %% m/mercator
```

```
FRAME_STEP_MT = 1/3;        %% Step (in meters) in the distance from the
hub used to determine each frame time
PANORAMA_STEP_MT = 8;       %% Step (in meters) in the distance from the
hub used to determine each panorama time
PANORAMA_OFFSET = 20;       %% Approx. distance between the center of
the first/last panorama and the origin/end of the segment

%%% List of files to be processed
filename{1} = 'June27GPS1';
filename{2} = 'June27GPS2';
filename{3} = 'June27GPS3';
filename{4} = 'June27GPS4';
filename{5} = 'June29GPS1';
filename{6} = 'June29GPS2';
filename{7} = 'June29GPS3';
filename{8} = 'June29GPS4';
filename{9} = 'June29GPS5';
filename{10} = 'July1GPS1';
filename{11} = 'July1GPS2';
filename{12} = 'July1GPS3';
filename{13} = 'July1GPS5';
filename{14} = 'July1GPS6';
filename{15} = 'July3GPS1';
filename{16} = 'July3GPS2';
filename{17} = 'July3GPS3';


ColorArray = ['r' 'g'];
%% Set the axis to convert the input coordinates from geographic to
mercator
figure(1);clf;
axesm('MapProjection','mercator');

%%% Load the calibration file
load G:\users\dibe\GPSData\matlab\svFileListCalib.txt

%%% Loading the hub splines
load G:\users\dibe\GPSData\matlab\Colorado2Spline traj_sp;
Colorado_spline = traj_sp;
load G:\users\dibe\GPSData\matlab\FairOaks2Spline traj_sp;
FairOaks_spline = traj_sp;


%%% Loading all the processed data
load G:\users\dibe\GPSData\matlab\PasadenaAllData Segment Street
Intersection ESRIIntersection Turns

%% Fixing a wrong street name label
Segment(879).StreetName = 'COLORADO';


% Initially we will only use segments that are contained in sequences
[5,6,7,10]

%% Find the segments that belong to sequences [5,6,7,10], have been
labeled with a StreetName and Numbers
```

```
segmSeqIndex = cat(1,Segment.Sequence);
allFromInt = cat(1,Segment.FromIntersection);
allToInt = cat(1,Segment.ToIntersection);
allFirstAddress = zeros(length(Segment),1);
for i = 1:length(Segment)
   if (~isempty(Segment(i).FirstAddress))
      allFirstAddress(i) = Segment(i).FirstAddress;
   end;
end;

segmInd = find((allFirstAddress ~= 0)&((segmSeqIndex ==
5)|(segmSeqIndex == 6)| ...
                                        (segmSeqIndex ==
7)|(segmSeqIndex == 10)));

alreadyPicked = zeros(1,length(Segment));

curr_index = 1;

lastLoadedSeq = 0;
for segm = segmInd'

   if (alreadyPicked(segm) == 0)

      FromInt = Segment(segm).FromIntersection;
      ToInt = Segment(segm).ToIntersection;

      %% Find all other segments with same FromInt and ToInt
      equivSegm = find((allFromInt == FromInt) & (allToInt == ToInt));

      if (length(equivSegm) == 1)
         BestSegm = equivSegm;
      else
         BestSegm = 0;
         %% Choose the best of this segments according to some criteria
(speed, from turn, to turn)
         %% This is tricky because I would rather not pick segments
that start or end with a left turn
         %% For the sake of simplicity let's pick the first segment
that comes from a turn and ends in a turn
         %% (hopefully a right turn)
         for i = 1:length(equivSegm)
            if ((Segment(equivSegm(i)).FromTurn ~=
0)&(Segment(equivSegm(i)).ToTurn ~= 0))
               BestSegm = equivSegm(i);
               break;
            end;
         end;
         %% if there are no segments that start with a turn and end
with a turn just pick the first one in the list
         if (BestSegm == 0)
            BestSegm = equivSegm(1);
         end;
      end;
      %% Add all these segments to the alreadyPicked list
      alreadyPicked(equivSegm) = 1;
```

```
      %% Determing if the segment contains odd or even numbers
      if (rem(Segment(BestSegm).FirstAddress,2) == 0)
         FirstSegmPos = 1;
      else
         FirstSegmPos = 2;
      end;
      ThisSegm.SegmIndex(1,FirstSegmPos) = BestSegm;
      EvenOddSegm(FirstSegmPos) = BestSegm;

      %% Find all segments that go from ToInt to FromInt
      equivSegm = find((allFromInt == ToInt) & (allToInt == FromInt));

      if (isempty(equivSegm))
         BestSegm = NaN;
      elseif (length(equivSegm) == 1)
         BestSegm = equivSegm;
      else
         BestSegm = 0;
         %% Choose the best of this segments according to some criteria
(speed, from turn, to turn)
         %% This is tricky because I would rather not pick segments
that start or end with a left turn
         %% For the sake of simplicity let's pick the first segment
that comes from a turn and ends in a turn
         %% (hopefully a right turn)
         for i = 1:length(equivSegm)
            if ((Segment(equivSegm(i)).FromTurn ~=
0)&(Segment(equivSegm(i)).ToTurn ~= 0))
               BestSegm = equivSegm(i);
               break;
            end;
         end;
         %% if there are no segments that start with a turn and end
with a turn just pick the first one in the list
         if (BestSegm == 0)
            BestSegm = equivSegm(1);
         end;
      end;
      %% Add all these segments to the alreadyPicked list
      alreadyPicked(equivSegm) = 1;

      ThisSegm.SegmIndex(1,(~(FirstSegmPos-1)+1)) = BestSegm;
      EvenOddSegm(~(FirstSegmPos-1)+1) = BestSegm;

      %%
      if (~isnan(BestSegm))
         if (strcmp(Segment(EvenOddSegm(1)).StreetName,
Segment(EvenOddSegm(2)).StreetName) == 0)
            disp('mmh, very weird. The even and odd segments have
different street names !');
            keyboard;
         end;
      end;

      ThisSegm.StreetName =
Segment(EvenOddSegm(FirstSegmPos)).StreetName;
```

```
      %% Process the two segments to determine the remaining field
entries
      for j = 1:2
         segm_ind = EvenOddSegm(j);
         if (~isnan(segm_ind))
            %%% Load the traj_sp and the other info about the sequence
            if (Segment(segm_ind).Sequence ~= lastLoadedSeq)
                seqname = filename{Segment(segm_ind).Sequence};
                eval(['load
G:\users\dipe\GPSData\matlab\',seqname,'Turns coord traj_sp curvC;']);
                FirstSampleUTCTime = coord(1,1);
                lastLoadedSeq = Segment(segm_ind).Sequence;

                figure(1);
                [coordX,coordY] = mfwdtran(coord(:,2),coord(:,3));
                coordX = coordX - ORIGIN_X;
                coordY = coordY - ORIGIN_Y;
                [coordX,coordY,Samples_Ind] = cleanUpGPSData(coordX,
coordY, MIN_SAMPLES_DIST);
                coordT = coord(Samples_Ind,1);
                al_coord = fnbrk(traj_sp,'breaks');  %% Accumulated
square root of cord-length

                %time_sp = csapi(al_coord,coordT);
                time_sp = spaps(al_coord,coordT,0.1,1); %% Linear
interpolation with tolerance 0.1 sec
            end;

            %%% Compute the extended coordinates
            %% Extending the arc lenght interval of the segment to the
center of the adjacent turn
            FromTurn = Segment(segm_ind).FromTurn;
            if (FromTurn > 0)
               Segment(segm_ind).ExtStart = Turns(FromTurn).CornerInd;
            else
               Segment(segm_ind).ExtStart = Segment(segm_ind).Start;
            end;
            ToTurn = Segment(segm_ind).ToTurn;
            if (ToTurn > 0)
                Segment(segm_ind).ExtStop = Turns(ToTurn).CornerInd;
            else
               Segment(segm_ind).ExtStop = Segment(segm_ind).Stop;
            end;


            %%% Need to determine the Hub side (N,S,W,E)
            dir = fnval(traj_sp,curvC(Segment(segm_ind).Stop)) -
fnval(traj_sp,curvC(Segment(segm_ind).Start));
            dir = dir/norm(dir);
            AlStart = curvC(Segment(segm_ind).ExtStart);
            AlStop  = curvC(Segment(segm_ind).ExtStop );
            StartCoord = fnval(traj_sp,AlStart);
            StopCoord = fnval(traj_sp,AlStop);
            MedCoord = (StartCoord + StopCoord)/2;

            %figure(2);
            %plotCoord = fnval(traj_sp,[AlStart:0.001:AlStop]);
```

```
   plot(SCALE*plotCoord(1,:),SCALE*plotCoord(2,:),ColorArray(j));
            drawnow;

            if ((dir(2) == 0) | (abs(dir(1)/dir(2)) > 1))
               %%% It's a Blvd (runs W-E). Look on which side of Fair
Oaks
               if (MedCoord(1) > -0.00044103)
                  HubSide = 'E';
               else
                  HubSide = 'W';
               end;
            else
               %%% It's an Ave (runs N-S). Look on which side of
COlorado
               if (MedCoord(2) > -0.00029266)
                   HubSide = 'N';
               else
                  HubSide = 'S';
               end;
            end;

            if ((HubSide == 'N')|(HubSide == 'S'))
               %%% Determine the HubCoord, coordinates of the point on
the Hub with the same x coord of MedCoord
               AL_interval = fnbrk(Colorado_spline,'interval');
               AL1 =
fmin('HorizDistEval',0,AL_interval(2),[],Colorado_spline,MedCoord(1));
               HubCoord = fnval(Colorado_spline,AL1);

               %%% Determine the local vert coords (in meters) of the
frames and panoramas
               SegmLength_mt = abs(StartCoord(2) - StopCoord(2))*SCALE;
               FrameLocalCoord_mt = [0:FRAME_STEP_MT:SegmLength_mt];
               NumFrames = length(FrameLocalCoord_mt);
               NumPanoramas = round((SegmLength_mt -
2*PANORAMA_OFFSET)/PANORAMA_STEP_MT) + 1;
               if (NumPanoramas <= 0)
                  PanoramaLocalCoord_mt = SegmLength_mt/2;
                  NumPanoramas = 1;
               else
                  PanoramaLocalCoord_mt = [0:1:(NumPanoramas-
1)]*PANORAMA_STEP_MT + ...
                                          (SegmLength_mt - (NumPanoramas-
1)*PANORAMA_STEP_MT)/2;
               end;

               %%% Determine the absolute mercator vert coordinates of
the frames and panoramas
               SegmOrient = sign(StopCoord(2) - StartCoord(2));
               if ((SegmOrient == 1)&(HubSide == 'N'))
                  PanoramaCoord = StartCoord(2) +
PanoramaLocalCoord_mt/SCALE;
                  FrameCoord = StartCoord(2) +
FrameLocalCoord_mt/SCALE;
                  DistanceFromHub = SCALE*abs(StartCoord(2) -
HubCoord(2));
```

```
                elseif ((SegmOrient == -1)&(HubSide == 'N'))
                   PanoramaCoord = StopCoord(2) +
PanoramaLocalCoord_mt/SCALE;
                   FrameCoord = StopCoord(2) + FrameLocalCoord_mt/SCALE;
                   DistanceFromHub = SCALE*abs(StopCoord(2) -
HubCoord(2));
                elseif ((SegmOrient == 1)&(HubSide == 'S'))
                   PanoramaCoord = StopCoord(2) -
PanoramaLocalCoord_mt/SCALE;
                   FrameCoord = StopCoord(2) - FrameLocalCoord_mt/SCALE;
                   DistanceFromHub = SCALE*abs(StopCoord(2) -
HubCoord(2));
                elseif ((SegmOrient == -1)&(HubSide == 'S'))
                   PanoramaCoord = StartCoord(2) -
PanoramaLocalCoord_mt/SCALE;
                   FrameCoord = StartCoord(2) -
FrameLocalCoord_mt/SCALE;
                   DistanceFromHub = SCALE*abs(StartCoord(2) -
HubCoord(2));
                end;
                %% Define a first guess of the offset
                Offset = 19.05;  % meters (this is the offset that I
have observed in the vertical addresses
                %% For each panorama and frame vert coord, find the
corresponding time in the sequence
                for i = 1:NumFrames
                   AlTarget =
fmin('VertDistEval',AlStart,AlStop,[],traj_sp,FrameCoord(i));
                   FrameUTCTime(i,1) = fnval(time_sp, AlTarget);
                end;
                for i = 1:NumPanoramas
                   AlTarget =
fmin('VertDistEval',AlStart,AlStop,[],traj_sp,PanoramaCoord(i));
                   PanoramaUTCTime(i,1) = fnval(time_sp, AlTarget);
                end;


             elseif ((HubSide == 'E')|(HubSide == 'W'))
                %%% Determine the HubCoord, coordinates of the point on
the Hub with the same y coord of MedCoord
                AL_interval = fnbrk(FairOaks_spline,'interval');
                AL1 =
fmin('VertDistEval',0,AL_interval(2),[],FairOaks_spline,MedCoord(2));
                HubCoord = fnval(FairOaks_spline,AL1);

                %%% Determine the local horiz coords (in meters) of the
frames and panoramas
                SegmLength_mt = abs(StartCoord(1) - StopCoord(1))*SCALE;
                FrameLocalCoord_mt = [0:FRAME_STEP_MT:SegmLength_mt];
                NumFrames = length(FrameLocalCoord_mt);
                NumPanoramas = round((SegmLength_mt -
2*PANORAMA_OFFSET)/PANORAMA_STEP_MT) + 1;
                if (NumPanoramas <= 0)
                   PanoramaLocalCoord_mt = SegmLength_mt/2;
                   NumPanoramas = 1;
                else
```

```
                PanoramaLocalCoord_mt = [0:1:(NumPanoramas-
1)]*PANORAMA_STEP_MT + ...
                                        (SegmLength_mt - (NumPanoramas-
1)*PANORAMA_STEP_MT)/2;
             end;

             %%% Determine the absolute mercator vert coordinates of
the frames and panoramas
             SegmOrient = sign(StopCoord(1) - StartCoord(1));
             if ((SegmOrient == 1)&(HubSide == 'E'))
                PanoramaCoord = StartCoord(1) +
PanoramaLocalCoord_mt/SCALE;
                FrameCoord = StartCoord(1) +
FrameLocalCoord_mt/SCALE;
                DistanceFromHub = SCALE*abs(StartCoord(1) -
HubCoord(1));
             elseif ((SegmOrient == -1)&(HubSide == 'E'))
                PanoramaCoord = StopCoord(1) +
PanoramaLocalCoord_mt/SCALE;
                FrameCoord = StopCoord(1) + FrameLocalCoord_mt/SCALE;
                DistanceFromHub = SCALE*abs(StopCoord(1) -
HubCoord(1));
             elseif ((SegmOrient == 1)&(HubSide == 'W'))
                PanoramaCoord = StopCoord(1) -
PanoramaLocalCoord_mt/SCALE;
                FrameCoord = StopCoord(1) - FrameLocalCoord_mt/SCALE;
                DistanceFromHub = SCALE*abs(StopCoord(1) -
HubCoord(1));
             elseif ((SegmOrient == -1)&(HubSide == 'W'))
                PanoramaCoord = StartCoord(1) -
PanoramaLocalCoord_mt/SCALE;
                FrameCoord = StartCoord(1) -
FrameLocalCoord_mt/SCALE;
                DistanceFromHub = SCALE*abs(StartCoord(1) -
HubCoord(1));
             end;
             %% Define a first guess of the offset
             Offset = 0.0;  % meters (this is the offset that I have
observed in the horizontal addresses

             %% For each panorama and frame vert coord, find the
corresponding time in the sequence
             for i = 1:NumFrames
                AlTarget =
fmin('HorizDistEval',AlStart,AlStop,[],traj_sp,FrameCoord(i));
                FrameUTCTime(i,1) = fnval(time_sp, AlTarget);
             end;
             for i = 1:NumPanoramas
                AlTarget =
fmin('HorizDistEval',AlStart,AlStop,[],traj_sp,PanoramaCoord(i));
                PanoramaUTCTime(i,1) = fnval(time_sp, AlTarget);
             end;

          end; %if ((HubSide == 'N')|(HubSide == 'S'))

          %% This part is common to the 'N','S' and the 'W','E' case
```

```
            % Compute the video time from the GPS sequence UTC time
using the calibration
            FrameVTime = FrameUTCTime - FirstSampleUTCTime +
svFileListCalib(Segment(segm_ind).Sequence);
            PanoramaVTime = PanoramaUTCTime - FirstSampleUTCTime +
svFileListCalib(Segment(segm_ind).Sequence);


            %% Save the results in ThisSegm recorc
            ThisSegm.DistanceFromHub(1,j) = DistanceFromHub;
            ThisSegm.Length(1,j) = SegmLength_mt;
            ThisSegm.Offset(1,j) = Offset;
            ThisSegm.OffsetFlag(1,j) = 0;
            ThisSegm.NumPanoramas(1,j) = NumPanoramas;
            ThisSegm.PanoramaCoord(1:NumPanoramas,j) =
PanoramaLocalCoord_mt;
            ThisSegm.PanoramaTime(1:NumPanoramas,j) = PanoramaVTime;
            ThisSegm.NumFrames(1,j) = NumFrames;
            ThisSegm.FrameTime(1:NumFrames,j) = FrameVTime;

            ThisSegm.SideofHub = HubSide;
            ThisSegm.FrameSeq(1,j) = Segment(segm_ind).Sequence;

            ThisSegm.FromAddress(1,j) = round((DistanceFromHub -
Offset)/3.81)*2 + (j-1);
            ThisSegm.ToAddress(1,j) = round((DistanceFromHub +
SegmLength_mt - Offset)/3.81)*2 + (j-1);

            clear FrameUTCTime PanoramaUTCTime
         end; % if (~isnan(segm_ind))
      end; %% for j = 1:2

      %% Add the new entry to the array
      SegmArray(curr_index) = ThisSegm;
      disp(num2str(curr_index));
      curr_index = curr_index + 1;

      clear ThisSegm
   end;
end;


%save G:\users\dibe\GPSdata\matlab\SegmArray_270800 SegmArray


%for segm = 1:length(SegmArray)
%   NPan(segm,:) = SegmArray(segm).NumPanoramas;
%end;

%break;

FillInNavigationFields;  %% Computed the rest of the fields and save
the result


%save G:\users\dibe\GPSdata\matlab\SegmArray_030900 SegmArray
```

```
Filename: FillinNavigationFields.m
%% Fill in the FromSegment and ToSegment fields
%%% and the Mercator and geo coordinates of the two sides of the
segment

%%% This script should be called after calling GenSegmArray
%%% EDB300800

%load G:\users\dibe\GPSdata\matlab\SegmArray_270800
%%% Loading all the processed data
load \\picolit\G$\users\dibe\GPSData\matlab\PasadenaAllData Segment
Street Intersection ESRIIntersection Turns

GPSSegmentInd = zeros(length(SegmArray),2);
for segm = 1:length(SegmArray)
   GPSSegmentInd(segm,:) = SegmArray(segm).SegmIndex;
end;

%for segm = [4 10 13 16:length(SegmArray)]
for segm = 256:length(SegmArray)

   [SegmIndex, j] = min(SegmArray(segm).SegmIndex);

   %% Determine the coordinates of the start and stop of the
corresponding ESRI street segment
      ESRIind = Segment(SegmIndex).FeatureElInds;
      FromAddr = cat(1,Street(ESRIind).R_F_ADDR);
      [tmpval,sort_ind] = sort(FromAddr);

      SegmArray(segm).FromMapCoordGeo =
[Street(ESRIind(sort_ind(1))).startLatitude;
Street(ESRIind(sort_ind(1))).startLongitude];
      SegmArray(segm).ToMapCoordGeo =
[Street(ESRIind(sort_ind(end))).stopLatitude;
Street(ESRIind(sort_ind(end))).stopLongitude];

      figure(1);
      [coordX,coordY] =
mfwdtran(SegmArray(segm).FromMapCoordGeo(1),SegmArray(segm).FromMapCoor
dGeo(2));
      SegmArray(segm).FromMapCoordMerc(1,1) = coordX - ORIGIN_X;
      SegmArray(segm).FromMapCoordMerc(2,1) = coordY - ORIGIN_Y;
      [coordX,coordY] =
mfwdtran(SegmArray(segm).ToMapCoordGeo(1),SegmArray(segm).ToMapCoordGeo
(2));
      SegmArray(segm).ToMapCoordMerc(1,1) = coordX - ORIGIN_X;
      SegmArray(segm).ToMapCoordMerc(2,1) = coordY - ORIGIN_Y;
   %% done
end;


for segm = 256:length(SegmArray)

   [SegmIndex, j] = min(SegmArray(segm).SegmIndex);


   if (Segment(SegmIndex).FeatureElOrient == 1)
```

```
      IntIndex(1) = Segment(SegmIndex).FromIntersection;
      IntIndex(2) = Segment(SegmIndex).ToIntersection;
   else
      IntIndex(1) = Segment(SegmIndex).ToIntersection;
      IntIndex(2) = Segment(SegmIndex).FromIntersection;
   end;

   for i = 1:2

      %%% All GPS segments that converge to this intersection
      segmList = Intersection(IntIndex(i)).Segments;

      StreetSegmList = [];
      %%% Find all street segments that contain this GPS segments
      for k = 1:length(segmList)
         [tmprow,tmpcol] = find(GPSSegmentInd == segmList(k));
         if (~isempty(tmprow))
            StreetSegmList = union(StreetSegmList,tmprow);
         end;
      end;

      %%% Sort the StreetSegmList accorcing to geographical direction
from the intersection
      VertSegm = [];
      HorizSegm = [];
      for k = 1:length(StreetSegmList)
         if ((SegmArray(StreetSegmList(k)).SideofHub ==
'N')|(SegmArray(StreetSegmList(k)).SideofHub == 'S'))
            VertSegm = [VertSegm StreetSegmList(k)];
         else
            HorizSegm = [HorizSegm StreetSegmList(k)];
         end;
      end;

      IntCoord = Intersection(IntIndex(i)).coord;

      DirSegmArray = zeros(4,1); %'E' 'N' 'W' 'S'
      if (~isempty(VertSegm))
         for s = 1:length(VertSegm)
            SegmCoord = [SegmArray(VertSegm(s)).FromMapCoordMerc(2,1)
SegmArray(VertSegm(s)).ToMapCoordMerc(2,1)];
            [absdist,tmpind] = max(abs(SegmCoord - IntCoord(2)));
            if ((SegmCoord(tmpind(1)) - IntCoord(2)) > 0.0)
               DirSegmArray(2) = VertSegm(s);
            else
               DirSegmArray(4) = VertSegm(s);
            end;
         end;
      end;

      if (~isempty(HorizSegm))
         for s = 1:length(HorizSegm)
            SegmCoord = [SegmArray(HorizSegm(s)).FromMapCoordMerc(1,1)
SegmArray(HorizSegm(s)).ToMapCoordMerc(1,1)];
            [absdist,tmpind] = max(abs(SegmCoord - IntCoord(1)));
            if ((SegmCoord(tmpind(1)) - IntCoord(1)) > 0.0)
               DirSegmArray(1) = HorizSegm(s);
```

```
            else
               DirSegmArray(3) = HorizSegm(s);
            end;
         end;
      end;


      if (i == 1)
         SegmArray(segm).FromSegmentsList = StreetSegmList;
         SegmArray(segm).FromSegments = DirSegmArray;
      else
         SegmArray(segm).ToSegmentsList = StreetSegmList;
         SegmArray(segm).ToSegments = DirSegmArray;
      end;


   end;
end;


%save G:\users\dibe\GPSdata\matlab\SegmArray_300800 SegmArray
```

```
Filename: ComputePanoramas.m
% Compute Panoramas.m

%% THis script iterated through the SegmArray list and, for each
segments, computes the even and odd side
%% panoramas using the GenPanorama.exe application.

%load G:\users\dibe\GPSdata\matlab\SegmArray_270800
%load G:\users\dibe\GPSdata\matlab\SegmArray_030900
load G:\users\dibe\GPSdata\matlab\SegmArray_100900

ImageQuality = 50; % 100 -> best quality

%for segm = [4 10 13 16:length(SegmArray)]
for segm = 618:length(SegmArray)

   for j = 1:2

      if (~isnan(SegmArray(segm).SegmIndex(1,j)))

         NumPanoramas = SegmArray(segm).NumPanoramas(1,j);

         for i = 1:NumPanoramas

            SeqTime = SegmArray(segm).PanoramaTime(i,j);
            SeqNumber = SegmArray(segm).FrameSeq(1,j) - 1;  % In the
basic program sequences start from 0

            FileName = ['Seg' num2str(segm) '_' num2str(j-1) '_D'
num2str(round(SegmArray(segm).PanoramaCoord(i,j)*100)) '.jpg'];

            disp(['... generating image ' FileName ]);

            eval(['! G:\users\dibe\VB\ComputePanorama\GenPanorama.exe '
num2str(SeqNumber) ' ' num2str(SeqTime) ' ' ...
                  FileName ' ' num2str(ImageQuality)]);

         end;

      end;

   end;

end;
```

```
Filename: ComputeFrames.m
%% THis script iterated through the SegmArray list and, for each
segments, computes the even and odd side
%% panoramas using the GenPanorama.exe application.

%load G:\users\dibe\GPSdata\matlab\SegmArray_270800
%load G:\users\dibe\GPSdata\matlab\SegmArray_030900
load G:\users\dibe\GPSdata\matlab\SegmArray_100900

ImageQuality = 50; % 100 -> best quality
FRAME_STEP_MT = 1/3;

%for segm = [4 10 13 16:length(SegmArray)]
for segm = 1:length(SegmArray)

   for j = 1:2

      if (~isnan(SegmArray(segm).SegmIndex(1,j)))

         NumFrames = SegmArray(segm).NumFrames(1,j);

         for i = 1:12:NumFrames

            SeqTime = SegmArray(segm).FrameTime(i,j);
            SeqNumber = SegmArray(segm).FrameSeq(1,j) - 1;  % In the
basic program sequences start from 0

            FileName = ['Seg' num2str(segm) '_' num2str(j-1) '_D'
num2str(round((i-1)*FRAME_STEP_MT*100)) '_Fr.jpg'];

            disp(['... generating image ' FileName ]);

            eval(['!
G:\users\dibe\VB\ComputeSingleFrame\GenSingleFrame.exe '
num2str(SeqNumber) ' ' num2str(SeqTime) ' ' ...
                  FileName ' ' num2str(ImageQuality)]);

         end;

      end;

   end;

end;
```

```
Filename: GenerateStreetSegmentDataDB.m
  SegmentIndex
% StreetName          String                  1x1
% FromAddress          Int                    1x2     [Even,Odd]
% ToAddress            Int                    1x2
% SideofHub           String                  1x1    One of the
following: 'N','S','W','E'
%
% FromMapCoordGeo     Double                  2x1  [lat,long]'
% FromMapCoordMerc    Double                  2x1  [x, y]'
% ToMapCoordGeo       Double                  2x1
% ToMapCoordMerc      Double                  2x1
%
% FromSegment           Int                   [4x1] In order: E,N,W,S
% ToSegment             Int                   [4x1]
%
% DistanceFromHub     Double                  [1x2]
% Length              Double                  [1x2]
% Offset              Double                  [1x2]
% OffsetFlag            Int                   [1x2]

%%% Open the database
logintimeout(5);
DBconn = database('segmentDB','','');
%DBcolnames =
{'SideofHub','StreetName','StreetNumber','SequenceNumber','VideoTime'};
DBcolnames =
{'SegmentIndex','StreetName','SideofHub','FromMapCoordLat','FromMapCoor
dLong','FromMapCoordX','FromMapCoordY',...

'ToMapCoordLat','ToMapCoordLong','ToMapCoordX','ToMapCoordY','FromSegme
ntE','FromSegmentN','FromSegmentW','FromSegmentS',...

'ToSegmentE','ToSegmentN','ToSegmentW','ToSegmentS','DistanceFromHubEve
n','DistanceFromHubOdd','LengthEven','LengthOdd',...

'OffsetEven','OffsetOdd','OffsetFlagEven','OffsetFlagOdd','NumPanoramas
Even','NumPanoramasOdd','NumFramesEven','NumFramesOdd',...
      'FrameSequenceEven','FrameSequenceOdd'};
Entry = cell(1,33);


%% Load Street Segment Array
%load G:\users\dibe\GPSdata\matlab\SegmArray_300800 SegmArray
%load G:\users\dibe\GPSdata\matlab\SegmArray_030900 SegmArray
load G:\users\dibe\GPSdata\matlab\SegmArray_100900 SegmArray


for segm = 1:length(SegmArray)

   disp(['Creating entry ' num2str(segm) ' in database']);

   %%% Write entries that are common to even and odd side
   Entry{1,1} = segm;
   Entry{1,2} = SegmArray(segm).StreetName ;
   Entry{1,3} = SegmArray(segm).SideofHub ;
   Entry{1,4} = SegmArray(segm).FromMapCoordGeo(1,1) ;
```

```
Entry{1,5} = SegmArray(segm).FromMapCoordGeo(2,1) ;
Entry{1,6} = SegmArray(segm).FromMapCoordMerc(1,1) ;
Entry{1,7} = SegmArray(segm).FromMapCoordMerc(2,1) ;
Entry{1,8} = SegmArray(segm).ToMapCoordGeo(1,1) ;
Entry{1,9} = SegmArray(segm).ToMapCoordGeo(2,1) ;
Entry{1,10} = SegmArray(segm).ToMapCoordMerc(1,1) ;
Entry{1,11} = SegmArray(segm).ToMapCoordMerc(2,1) ;
Entry{1,12} = SegmArray(segm).FromSegments(1,1) ;
Entry{1,13} = SegmArray(segm).FromSegments(2,1) ;
Entry{1,14} = SegmArray(segm).FromSegments(3,1) ;
Entry{1,15} = SegmArray(segm).FromSegments(4,1) ;
Entry{1,16} = SegmArray(segm).ToSegments(1,1) ;
Entry{1,17} = SegmArray(segm).ToSegments(2,1) ;
Entry{1,18} = SegmArray(segm).ToSegments(3,1) ;
Entry{1,19} = SegmArray(segm).ToSegments(4,1) ;

%%% Write entries relative to even side
if (~isnan(SegmArray(segm).SegmIndex(1,1)))
   Entry{1,20} = SegmArray(segm).DistanceFromHub(1,1) ;
   Entry{1,22} = SegmArray(segm).Length(1,1) ;
   Entry{1,24} = SegmArray(segm).Offset(1,1) ;
   Entry{1,26} = SegmArray(segm).OffsetFlag(1,1) ;
   Entry{1,28} = SegmArray(segm).NumPanoramas(1,1) ;
   Entry{1,30} = SegmArray(segm).NumFrames(1,1) ;
   Entry{1,32} = SegmArray(segm).FrameSeq(1,1) ;
else
   Entry{1,20} = 0;
   Entry{1,22} = 0;
   Entry{1,24} = 0;
   Entry{1,26} = 0;
   Entry{1,28} = 0;
   Entry{1,30} = 0;
   Entry{1,32} = -1;
end;

%%% Write entries relative to odd side
if (~isnan(SegmArray(segm).SegmIndex(1,2)))
   Entry{1,21} = SegmArray(segm).DistanceFromHub(1,2) ;
   Entry{1,23} = SegmArray(segm).Length(1,2) ;
   Entry{1,25} = SegmArray(segm).Offset(1,2) ;
   Entry{1,27} = SegmArray(segm).OffsetFlag(1,2) ;
   Entry{1,29} = SegmArray(segm).NumPanoramas(1,2) ;
   Entry{1,31} = SegmArray(segm).NumFrames(1,2) ;
   Entry{1,33} = SegmArray(segm).FrameSeq(1,2) ;
else
   Entry{1,21} = 0 ;
   Entry{1,23} = 0 ;
   Entry{1,25} = 0 ;
   Entry{1,27} = 0 ;
   Entry{1,29} = 0 ;
   Entry{1,31} = 0 ;
   Entry{1,33} = -1 ;
end;

   insert(DBconn,'StreetSegmentData',DBcolnames,Entry);  %% Insert the
new entry in table
```

```
end;
close(DBconn)
```

```
Filename: GeneratePanoramaTableDB.m
%%: Open the database
logintimeout(5);
DBconn = database('segmentDB','','');
%DBcolnames =
{'SideofHub','StreetName','StreetNumber','SequenceNumber','VideoTime'};
DBcolnames = {'SegmentIndex','PanoramaDistance','EvenOdd'};
Entry = cell(1,3);


%% Load Street Segment Array
%load G:\users\dibe\GPSdata\matlab\SegmArray_300800 SegmArray
load G:\users\dibe\GPSdata\matlab\SegmArray_100900 SegmArray

for segm = 256:length(SegmArray)
   disp(['Creating entries for segment ' num2str(segm) ' in
database']);

   for j = 1:2

      if (~isnan(SegmArray(segm).SegmIndex(1,j)))

         for p = 1:SegmArray(segm).NumPanoramas(1,j)

            Entry{1,1} = segm;
            Entry{1,2} = round(SegmArray(segm).PanoramaCoord(p,j)*100);
            Entry{1,3} = j-1;

            insert(DBconn,'PanoramaData',DBcolnames,Entry);  %%% Insert
the new entry in table

         end;

      end;

   end;

end;

close(DBconn);
```

```
Filename: GenerateCoordsLookUpTableDB.m
%% Generate look up table Table in the Access Database

SCALE = 5272498.48904543;  % m/mercator


% Load Street Segment Array
%load G:\users\dibe\GPSdata\matlab\SegmArray_300800 SegmArray
load G:\users\dibe\GPSdata\matlab\SegmArray_100900 SegmArray


SegmFromCoords = cat(2,SegmArray.FromMapCoordMerc);
SegmToCoords = cat(2,SegmArray.ToMapCoordMerc);
Xint(1) = min([SegmFromCoords(1,:) SegmToCoords(1,:)]);
Xint(2) = max([SegmFromCoords(1,:) SegmToCoords(1,:)]);
Yint(1) = min([SegmFromCoords(2,:) SegmToCoords(2,:)]);
Yint(2) = max([SegmFromCoords(2,:) SegmToCoords(2,:)]);

BLOCK_SIZE = 0.5e-4;

%% We divice the map in square blocks having size 1e-4x1e-4 (~527x527
m).
%% The point P = (x,y) will belong to block (round(x*1e4),round(y*1e4))
%%
%% For each street Segment, we add the Segment index to all the square
blocks crossed by that segment
%% We build a look-up table CoordsLookUpTable that has the following
structure:
%% The first column contains the X labels and the second column the Y
labels of the block. Column 3 on contain all the segment
%% indexes of the segments that cross that block

CurrNumBlocks = 0;
CoordsLookUpTable = [];
for segm = 1:length(SegmArray)
   Xint = [min([SegmFromCoords(1,segm) SegmToCoords(1,segm)])
max([SegmFromCoords(1,segm) SegmToCoords(1,segm)]) ];
   Yint = [min([SegmFromCoords(2,segm) SegmToCoords(2,segm)])
max([SegmFromCoords(2,segm) SegmToCoords(2,segm)]) ];
   for xind =
round(Xint(1)/BLOCK_SIZE)*BLOCK_SIZE:BLOCK_SIZE:round(Xint(2)/BLOCK_SIZ
E)*BLOCK_SIZE
      for yind =
round(Yint(1)/BLOCK_SIZE)*BLOCK_SIZE:BLOCK_SIZE:round(Yint(2)/BLOCK_SIZ
E)*BLOCK_SIZE
         if ~isempty(CoordsLookUpTable)
            block_ind = intersect(find(abs(CoordsLookUpTable(:,1) -
xind) < 1e-16) , find(abs(CoordsLookUpTable(:,2) - yind) < 1e-16));
            if isempty(block_ind)
               %% create new entry
               new_entry = [xind yind segm];
               CoordsLookUpTable(CurrNumBlocks + 1,1:3) = new_entry;
               CurrNumBlocks = CurrNumBlocks + 1;
            else
               last_nonempty_ind =
max(find(CoordsLookUpTable(block_ind,3:end) ~= 0)) + 2;
               CoordsLookUpTable(block_ind,last_nonempty_ind+1) = segm;
```

```
            end;
         else
            CoordsLookUpTable = [xind yind segm];
         end;
      end;
   end;
end;


%%% Save the table in DB
%%% Open the database
logintimeout(5);
DBconn = database('segmentDB','','');
%DBcolnames =
{'SideofHub','StreetName','StreetNumber','SequenceNumber','VideoTime'};
DBcolnames = {'XCoordLabel','YCoordLabel','SegmentIndex'};
Entry = cell(1,3);

for i = 1:size(CoordsLookUpTable,1)
   j = 3;
   while ((j <= size(CoordsLookUpTable,2)) & (CoordsLookUpTable(i,j) ~=
0))
      Entry{1,1} = round(CoordsLookUpTable(i,1)*1e5);
      Entry{1,2} = round(CoordsLookUpTable(i,2)*1e5);
      Entry{1,3} = CoordsLookUpTable(i,j);

      insert(DBconn,'CoordsLookUpTableData',DBcolnames,Entry);  %%%
Insert the new entry in table

      j = j + 1;
   end;
end;

close(DBconn)
```

Appendix – VisualBasic and ASP code for the server side process

Filename: GetSegmentAndBusinessInfo.asp

```
<%@ LANGUAGE = VBScript %>


<%


SegmIndex = Request("SegmentIndex")


'Create a connection object
 Set cn = Server.CreateObject("ADODB.Connection")
'Open a connection; the string refers to the DSN
 cn.Open  "FILEDSN=segmentDBFile.dsn"

'Response.Write "Connection State = " & cn.State & "<BR>"

'Instantiate a Recordset object
 Set rsSegment = Server.CreateObject("ADODB.Recordset")

'Open a recordset using the Open method
' and use the connection established by the Connection object
'strSQL = "SELECT SideofHub, StreetName, StreetNumber, SequenceNumber, VideoTime
"
'& " FROM AddressTable WHERE StreetName = '" & Request("StreetName") & "'" _
'& " And StreetNumber = " & Request("StreetNumber") _
'& " And SideofHub = '" & Request("HubSide") & "'"

strSQL = "SELECT * FROM StreetSegmentData WHERE SegmentIndex = " & SegmIndex

'strSQL = "SELECT * FROM StreetSegmentData"

'Response.Write "SQL String:  " & strSQL & "<BR> <BR> <BR>"

rsSegment.Open  strSQL, cn

'Cycle through record set and display the results
' and increment record position with MoveNext method
'Set objStreetNumber = rsAddress("StreetNumber")
'Set objHubSide = rsAddress("SideofHub")
'Set objStreetName = rsAddress("StreetName")
'Set objSequenceNumber = rsAddress("SequenceNumber")
'Set objVideoTime = rsAddress("VideoTime")

'
Response.Write rsSegment("SegmentIndex")
```

```
Response.Write Chr(10)
Response.Write rsSegment("StreetName")
Response.Write Chr(10)
Response.Write rsSegment("SideofHub")
Response.Write Chr(10)
Response.Write rsSegment("FromMapCoordLat")
Response.Write Chr(10)
Response.Write rsSegment("FromMapCoordLong")
Response.Write Chr(10)
Response.Write rsSegment("FromMapCoordX")
Response.Write Chr(10)
Response.Write rsSegment("FromMapCoordY")
Response.Write Chr(10)
Response.Write rsSegment("ToMapCoordLat")
Response.Write Chr(10)
Response.Write rsSegment("ToMapCoordLong")
Response.Write Chr(10)
Response.Write rsSegment("ToMapCoordX")
Response.Write Chr(10)
Response.Write rsSegment("ToMapCoordY")
Response.Write Chr(10)
Response.Write rsSegment("FromSegmentE")
Response.Write Chr(10)
Response.Write rsSegment("FromSegmentN")
Response.Write Chr(10)
Response.Write rsSegment("FromSegmentW")
Response.Write Chr(10)
Response.Write rsSegment("FromSegmentS")
Response.Write Chr(10)
Response.Write rsSegment("ToSegmentE")
Response.Write Chr(10)
Response.Write rsSegment("ToSegmentN")
Response.Write Chr(10)
Response.Write rsSegment("ToSegmentW")
Response.Write Chr(10)
Response.Write rsSegment("ToSegmentS")
Response.Write Chr(10)
Response.Write rsSegment("DistanceFromHubEven")
Response.Write Chr(10)
Response.Write rsSegment("DistanceFromHubOdd")
Response.Write Chr(10)
Response.Write rsSegment("LengthEven")
Response.Write Chr(10)
Response.Write rsSegment("LengthOdd")
Response.Write Chr(10)
Response.Write rsSegment("OffsetEven")
Response.Write Chr(10)
Response.Write rsSegment("OffsetOdd")
Response.Write Chr(10)
Response.Write rsSegment("OffsetFlagEven")
Response.Write Chr(10)
Response.Write rsSegment("OffsetFlagOdd")
Response.Write Chr(10)
Response.Write rsSegment("NumPanoramasEven")
Response.Write Chr(10)
Response.Write rsSegment("NumPanoramasOdd")
Response.Write Chr(10)
```

```
Response.Write rsSegment("NumFramesEven")
Response.Write Chr(10)
Response.Write rsSegment("NumFramesOdd")
Response.Write Chr(10)
Response.Write rsSegment("FrameSequenceEven")
Response.Write Chr(10)
Response.Write rsSegment("FrameSequenceOdd")
Response.Write Chr(10)

Dim Offset(2)
Dim DistanceFromHub(2)
Dim Length(2)
Dim FromAddress(2)
Dim ToAddress(2)

NumPanoramasEven = rsSegment("NumPanoramasEven")
NumPanoramasOdd = rsSegment("NumPanoramasOdd")
StreetName = rsSegment("StreetName")
SideofHub = rsSegment("SideofHub")
Offset(0) = rsSegment("OffsetEven")
Offset(1) = rsSegment("OffsetOdd")
DistanceFromHub(0) = rsSegment("DistanceFromHubEven")
DistanceFromHub(1) = rsSegment("DistanceFromHubOdd")
Length(0) = rsSegment("LengthEven")
Length(1) = rsSegment("LengthOdd")

'Do Until rsAddress.EOF
'Response.Write "Found Entry: " & objStreetNumber & " " & objHubSide & " "_
'& objStreetName & " Sequence = " & objSequenceNumber & "  Video Time = " &
objVideoTime _
'& "<BR>"
'rsAddress.MoveNext
'Loop

rsSegment.Close

if (NumPanoramasEven > 0) Then

      'Query the PanoramaTable database fro the PanoramaCoords
      strSQL = "SELECT * FROM PanoramaData WHERE SegmentIndex = " & SegmIndex &
_
       " And EvenOdd = 0"

      rsSegment.Open  strSQL, cn

      Do Until rsSegment.EOF
            Response.Write rsSegment("PanoramaDistance")
            Response.Write Chr(10)
            rsSegment.MoveNext
      Loop
      rsSegment.Close
End if


if (NumPanoramasOdd > 0) Then
```

```
      'Query the PanoramaTable database fro the PanoramaCoords
      strSQL = "SELECT * FROM PanoramaData WHERE SegmentIndex = " & SegmIndex &
_
       " And EvenOdd = 1"

      rsSegment.Open  strSQL, cn

      Do Until rsSegment.EOF
            Response.Write rsSegment("PanoramaDistance")
            Response.Write Chr(10)
            rsSegment.MoveNext
      Loop
      rsSegment.Close
End if

cn.Close


'Now query the Businees Listing database to find all the businesses on this
segment
cn.Open  "FILEDSN=BusinessListingDB.dsn"

'Compute the first and last addresses on the two sides of the street segment
'EDB092000 Expanding the segment length by 24 m on both sides. It doesn't hurt
to get more businesses than
'to avoid some boundary effects.
FromAddress(0) = 2 * Round((DistanceFromHub(0) - Offset(0) - 24) / 3.81) - 0
FromAddress(1) = 2 * Round((DistanceFromHub(1) - Offset(1) - 24) / 3.81) - 1
ToAddress(0)   = 2 * Round((DistanceFromHub(0) + Length(0) - Offset(0) + 24) /
3.81) - 0
ToAddress(1)   = 2 * Round((DistanceFromHub(1) + Length(1) - Offset(1) + 24) /
3.81) - 1

if (FromAddress(0) <= FromAddress(1)) Then
      minFromAddress = FromAddress(0)
Else
      minFromAddress = FromAddress(1)
End if

if (ToAddress(0) >= ToAddress(1)) Then
      maxToAddress = ToAddress(0)
Else
      maxToAddress = ToAddress(1)
End if

'Response.Write minFromAddress & " " & maxToAddress

strSQL = "SELECT * FROM BusinessInfoTable WHERE StreetName = '" & StreetName &
"' AND SideOfHub = '" & SideofHub & "' AND StreetNumber >= " & minFromAddress &
_
" AND StreetNumber <= " & maxToAddress

'strSQL = "SELECT * FROM businessInfoTable WHERE StreetName = 'COLORADO'"

'Response.Write "<BR> " & strSQL & " <BR>"

rsSegment.Open  strSQL, cn
```

```
Do Until rsSegment.EOF
      StreetNumber = rsSegment("StreetNumber")
      if ( (((StreetNumber Mod 2) = 0) And (StreetNumber >= FromAddress(0)) And
(StreetNumber <= ToAddress(0))) Or _
           (((StreetNumber Mod 2) = 1) And (StreetNumber >= FromAddress(1)) And
(StreetNumber <= ToAddress(1))) ) Then
            Response.Write StreetNumber
            Response.Write Chr(10)
            Response.Write rsSegment("BusinessName")
            Response.Write Chr(10)
            Response.Write rsSegment("PhoneNumber")
            Response.Write Chr(10)
            Response.Write rsSegment("WebLink")
            Response.Write Chr(10)
      End if
      rsSegment.MoveNext
      'Response.Write "Next"
Loop
rsSegment.Close


%>
```

```
Filename: Address2Location.bas

'This CGI exe receives an address as input :
'
' StreetName, StreetAddress, SideofHub
'
'and returns the StreetSegment number that address is on and the coord in m
'of that address on the segment
'It returns StreetSegment = 0 if no match was found in the database

Option Explicit


Dim CGI As New CGI_Interface

Sub Main()

Dim StreetNumber As Integer
Dim StreetName As String
Dim SideofHub As String
Dim RunLocal As Integer
Dim SQLString As String
Dim StreetSegment As Integer
Dim FirstAddress As Integer
Dim LastAddress As Integer
Dim AddressSegmCoord As Double
Dim AddressFound As Boolean

CGI.InitCGI

RunLocal = 0
If RunLocal = 0 Then
    StreetNumber = CGI.GetCGI("StreetNumber")
    StreetName = CGI.GetCGI("StreetName")
    SideofHub = CGI.GetCGI("SideofHub")
Else
    StreetName = "colorado"
    StreetNumber = 101
    SideofHub = "E"
End If

 'Replace _ character with a space in StreetName String
StreetName = Replace(StreetName, "_", " ")

 Dim cn As ADODB.Connection
 Set cn = New Connection
'Open a connection; the string refers to the DSN
 cn.Open "FILEDSN=segmentDBFile.dsn"

 Dim rsSegment As ADODB.Recordset
 Set rsSegment = New Recordset

 SQLString = "SELECT * FROM StreetSegmentData WHERE StreetName = '" & _
 StreetName & "' And SideofHub = '" & SideofHub & "'"

 rsSegment.Open SQLString, cn
```

```
AddressFound = False
Do Until rsSegment.EOF
        Debug.Print "Segment: " & rsSegment("SegmentIndex")
        If ((StreetNumber Mod 2 = 0) And (rsSegment("FrameSequenceEven") <> -1))
Then    'Look in the even side of segment
            'round((DistanceFromHub - Offset)/3.81)*2 + (j-1);
            FirstAddress = Round((rsSegment("DistanceFromHubEven") - _
                        rsSegment("OffsetEven")) / 3.81) * 2
            LastAddress = Round((rsSegment("DistanceFromHubEven") + _
                        rsSegment("LengthEven") - _
                        rsSegment("OffsetEven")) / 3.81) * 2
            AddressSegmCoord = StreetNumber / 2 * 3.81 -
rsSegment("DistanceFromHubEven") + _
                        rsSegment("OffsetEven")
        ElseIf ((StreetNumber Mod 2 = 1) And (rsSegment("FrameSequenceOdd") <> -
1)) Then
            FirstAddress = Round((rsSegment("DistanceFromHubOdd") - _
                        rsSegment("OffsetOdd")) / 3.81) * 2 - 1
            LastAddress = Round((rsSegment("DistanceFromHubOdd") + _
                        rsSegment("LengthOdd") - _
                        rsSegment("OffsetOdd")) / 3.81) * 2 - 1
            AddressSegmCoord = (StreetNumber + 1) / 2 * 3.81 -
rsSegment("DistanceFromHubOdd") + _
                        rsSegment("OffsetOdd")
        End If

        If ((StreetNumber <= LastAddress) And (StreetNumber >= FirstAddress))
Then
            AddressFound = True
            StreetSegment = rsSegment("SegmentIndex")
            Debug.Print "Match found in Segment " & rsSegment("SegmentIndex")
            Debug.Print "First Address: " & FirstAddress
            Debug.Print "Last Address: " & LastAddress
            Debug.Print "Address Segment Coord: " & AddressSegmCoord
            Exit Do
        End If

        rsSegment.MoveNext
 Loop

 CGI.PrintCGI "Content-Type: text/html"
 CGI.PrintCGI ""


 If AddressFound Then
    CGI.PrintCGI StreetSegment & " " & AddressSegmCoord
    'CGI.PrintCGI AddressSegmCoord
 Else
    CGI.PrintCGI "0 0"
    'CGI.PrintCGI Chr(10)
    'CGI.PrintCGI "0"
 End If


End Sub
```

```
Filename: Coords2Segment.bas

'This CGI exe receives the mercator coordinates of a point as input
'
' XCoord, YCoord
'
'and returns the StreetSegment which is the closest to those coordinates,
'the side of the segment and the % of
'It returns StreetSegment = 0 if no match was found in the database

Option Explicit

Const BLOCK_SIZE = 0.00005
Const M_Merc_SCALE = 5272500#

Dim CGI As New CGI_Interface

Sub Main()

Dim XCoord As Double
Dim YCoord As Double
Dim XCoordLabel As Integer
Dim YCoordLabel As Integer
Dim RunLocal As Integer
Dim SQLString As String
Dim StreetSegment As Integer
Dim SegmentFound As Boolean

CGI.InitCGI

RunLocal = 0
If RunLocal = 0 Then
    XCoord = CGI.GetCGI("XCoord")
    YCoord = CGI.GetCGI("YCoord")
Else
    XCoord = -0.0003302291
    YCoord = -0.0003298909
End If

 Dim cn As ADODB.Connection
 Set cn = New Connection
'Open a connection; the string refers to the DSN
 cn.Open "FILEDSN=segmentDBFile.dsn"

 Dim rsSegment As ADODB.Recordset
 Set rsSegment = New Recordset

 XCoordLabel = Round(XCoord / BLOCK_SIZE) * BLOCK_SIZE * 100000#
 YCoordLabel = Round(YCoord / BLOCK_SIZE) * BLOCK_SIZE * 100000#

 SQLString = "SELECT * FROM StreetSegmentData,CoordsLookUpTableData WHERE
StreetSegmentData.SegmentIndex = CoordsLookUpTableData.SegmentIndex AND
XCoordLabel = "
 & XCoordLabel & " AND YCoordLabel = " & YCoordLabel

 rsSegment.Open SQLString, cn
```

```
Dim MinDist As Double
Dim MinSegm As Integer
Dim X1(2), X2(2), Un(2), V(2), Vn(2) As Double
Dim Dist, ODist, SegmCoord As Double
Dim SideofHub As String
SegmentFound = False
MinDist = 10000000000#
MinSegm = 0
Do Until rsSegment.EOF
        SegmentFound = True
        Debug.Print rsSegment("SegmentIndex"); "  "; rsSegment("StreetName")
        'Compute the distance between the segment and [XCoord YCoord]
        X1(1) = rsSegment("FromMapCoordX")
        X1(2) = rsSegment("FromMapCoordY")
        X2(1) = rsSegment("ToMapCoordX")
        X2(2) = rsSegment("ToMapCoordY")
        V(1) = X2(1) - X1(1)
        V(2) = X2(2) - X1(2)
        Vn(1) = V(1) / ((V(1) ^ 2 + V(2) ^ 2) ^ 0.5)
        Vn(2) = V(2) / ((V(1) ^ 2 + V(2) ^ 2) ^ 0.5)
        Un(1) = -Vn(2)
        Un(2) = Vn(1)
        Dist = (XCoord - X1(1)) * Un(1) + (YCoord - X1(2)) * Un(2)
        ODist = (XCoord - X1(1)) * Vn(1) + (YCoord - X1(2)) * Vn(2)
        If ((ODist >= 0) And (ODist <= (V(1) ^ 2 + V(2) ^ 2) ^ 0.5) And
(Abs(Dist) < Abs(MinDist))) Then
            'Update MinDist
            MinDist = Dist
            MinSegm = rsSegment("SegmentIndex")
            SegmCoord = ODist / ((V(1) ^ 2 + V(2) ^ 2) ^ 0.5)
            SideofHub = rsSegment("SideofHub")
        End If
        rsSegment.MoveNext
Loop

CGI.PrintCGI "Content-Type: text/html"
CGI.PrintCGI ""

If SegmentFound = False Then
   'There where no segments in the database that cross this squared block
   CGI.PrintCGI "0 0 0"
Else
   If MinSegm = 0 Then
       'This point is in none of the candidate segments' convex hull
       CGI.PrintCGI "0 0 0"
   Else
       Dim EvenOdd As Integer
       EvenOdd = (MinDist / Abs(MinDist) + 1) / 2
       'Fix the EvenOdd value
       If (SideofHub = "W") Or (SideofHub = "S") Then
           EvenOdd = 1 - EvenOdd
       End If
       CGI.PrintCGI MinSegm & " " & EvenOdd & " " & SegmCoord
   End If
End If
End Sub
```

Appendix - Code for extraction of video frames from an AVI file

Filename: Dump.cpp. Here reported only the part of the code that was modified by us. The original code was provided as a sample by Microsoft.

```
//
// Receive
//
// Do something with this media sample
//
STDMETHODIMP CDumpInputPin::Receive(IMediaSample *pSample)
{
    CAutoLock lock(m_pReceiveLock);
    PBYTE pbData;

    // Has the filter been stopped yet
    if (m_pDump->m_hFile == INVALID_HANDLE_VALUE) {
        return NOERROR;
    }

    REFERENCE_TIME tStart, tStop;
    pSample->GetTime(&tStart, &tStop);
    DbgLog((LOG_TRACE, 1, TEXT("tStart(%s), tStop(%s), Diff(%d ms), Bytes(%d)"),
        (LPCTSTR) CDisp(tStart),
        (LPCTSTR) CDisp(tStop),
        (LONG)((tStart - m_tLast) / 10000),
        pSample->GetActualDataLength()));

    m_tLast = tStart;

    // Copy the data to the file

    HRESULT hr = pSample->GetPointer(&pbData);
    if (FAILED(hr)) {
        return hr;
    }


//    return m_pDump->Write(pbData,pSample->GetActualDataLength());
            if (NumberofDownloadedColumns < PANORAMA_WIDTH){
                    while (abs(ColumnIndexes[NumberofDownloadedColumns]) ==
(RelFrameNumber+1)){
                            WriteFrameColumns(pSample,
(ColumnIndexes[NumberofDownloadedColumns] > 0), NumberofDownloadedColumns);
                            NumberofDownloadedColumns +=1;
                    }
                    RelFrameNumber += 1;
            }
            else if (NumberofDownloadedColumns == PANORAMA_WIDTH){
                    long param1 = 10;
                    long param2 = 20;
                    IFilterGraph * myFilterGraph = m_pDump->m_pFilter->GetFilterGraph();
                    IMediaEventSink * pMediaEventSink;
                    myFilterGraph->QueryInterface(IID_IMediaEventSink, (void **)&pMediaEventSink);
```

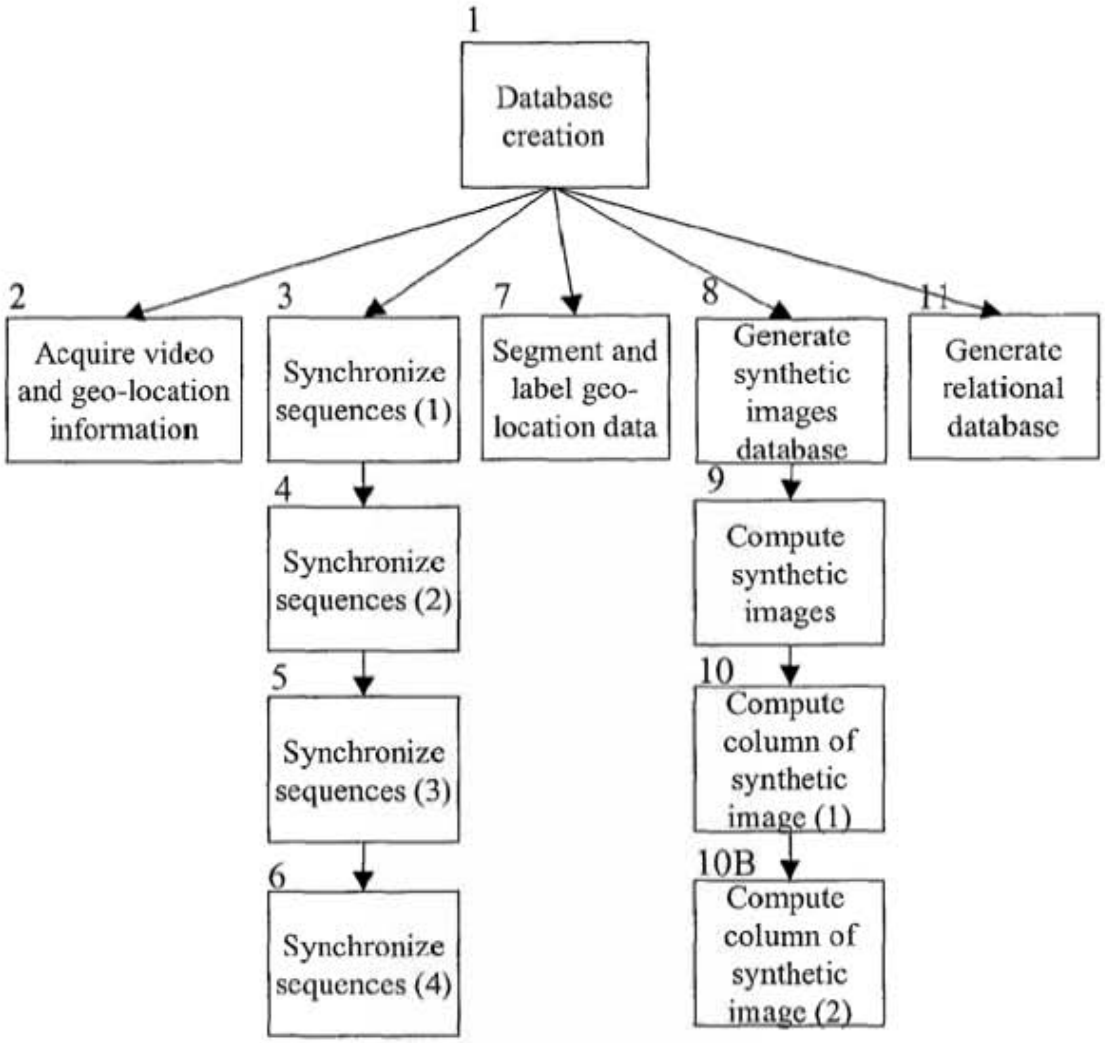
```
                pMediaEventSink->Notify(34567,0,0);
                //m_pDump->m_pFilter->NotifyEvent(34567,param1,param2);
        }

        return NOERROR;

//          return WriteStringInfo(pSample);
}


//
// DumpStringInfo
//
// Write to the file as text form
//
HRESULT CDumpInputPin::WriteStringInfo(IMediaSample *pSample)
{
    TCHAR TempString[256],FileString[256];
    PBYTE pbData;
        AM_MEDIA_TYPE *m_MediaType = NULL;


        // Retrieve media type
//      pSample->GetMediaType(&m_MediaType);
//      const GUID SubType = m_MediaType->subtype;

        //Write the subtype out
//    wsprintf(FileString,"\r\nSample subtype RGB565: (%d)",((SubType == MEDIASUBTYPE_RGB565)?
TRUE: FALSE));
//    m_pDump->WriteString(FileString);

    // Retrieve the time stamps from this sample

    REFERENCE_TIME tStart, tStop;
    pSample->GetTime(&tStart, &tStop);
    m_tLast = tStart;

    // Write the sample time stamps out

    wsprintf(FileString,"\r\nRenderer received sample (%dms)",timeGetTime());
    m_pDump->WriteString(FileString);
    wsprintf(FileString,"    Start time (%s)",CDisp(tStart));
    m_pDump->WriteString(FileString);
    wsprintf(FileString,"    End time (%s)",CDisp(tStop));
    m_pDump->WriteString(FileString);

    // Display the media times for this sample

    HRESULT hr = pSample->GetMediaTime(&tStart, &tStop);
    if (hr == NOERROR) {
        wsprintf(FileString,"    Start media time (%s)",CDisp(tStart));
        m_pDump->WriteString(FileString);
        wsprintf(FileString,"    End media time (%s)",CDisp(tStop));
        m_pDump->WriteString(FileString);
    }
```

```
// Is this a sync point sample

hr = pSample->IsSyncPoint();
wsprintf(FileString,"   Sync point (%d)",(hr == S_OK));
m_pDump->WriteString(FileString);

// Is this a preroll sample

hr = pSample->IsPreroll();
wsprintf(FileString,"   Preroll (%d)",(hr == S_OK));
m_pDump->WriteString(FileString);

// Is this a discontinuity sample

hr = pSample->IsDiscontinuity();
wsprintf(FileString,"   Discontinuity (%d)",(hr == S_OK));
m_pDump->WriteString(FileString);

// Write the actual data length

LONG DataLength = pSample->GetActualDataLength();
wsprintf(FileString,"   Actual data length (%d)",DataLength);
m_pDump->WriteString(FileString);

// Does the sample have a type change aboard

AM_MEDIA_TYPE *pMediaType;
pSample->GetMediaType(&pMediaType);
wsprintf(FileString,"   Type changed (%d)",
  (pMediaType ? TRUE : FALSE));
m_pDump->WriteString(FileString);
DeleteMediaType(pMediaType);

// Copy the data to the file

hr = pSample->GetPointer(&pbData);
if (FAILED(hr)) {
  return hr;
}

// Write each complete line out in BYTES_PER_LINES groups

//  for (int Loop = 0;Loop < (DataLength / BYTES_PER_LINE);Loop++) {
  for (int Loop = 0;Loop < (40 / BYTES_PER_LINE);Loop++) {
    wsprintf(FileString,FIRST_HALF_LINE,pbData[0],pbData[1],pbData[2],
          pbData[3],pbData[4],pbData[5],pbData[6],
            pbData[7],pbData[8],pbData[9]);
    wsprintf(TempString,SECOND_HALF_LINE,pbData[10],pbData[11],pbData[12],
          pbData[13],pbData[14],pbData[15],pbData[16],
            pbData[17],pbData[18],pbData[19]);
    lstrcat(FileString,TempString);
    m_pDump->WriteString(FileString);
    pbData += BYTES_PER_LINE;
  }
```

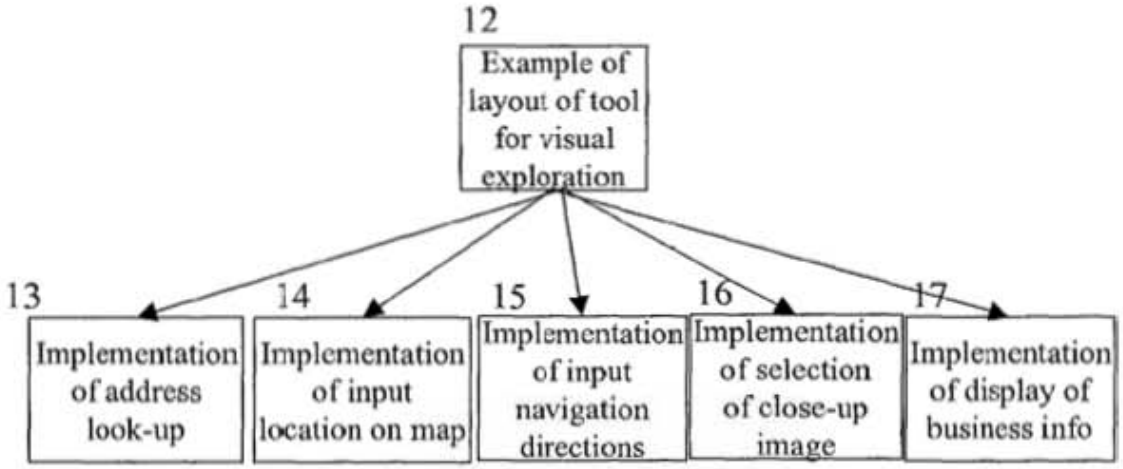
```
    // Write the last few bytes out afterwards

    wsprintf(FileString,"   ");
    for (Loop = 0;Loop < (DataLength % BYTES_PER_LINE);Loop++) {
      wsprintf(TempString,"%x ",pbData[Loop]);
      lstrcat(FileString,TempString);
    }
    m_pDump->WriteString(FileString);


        return NOERROR;
}

//
// DumpStringInfo
//
// Write to the file as text form
//
HRESULT CDumpInputPin::WriteFrameColumns(IMediaSample *pSample, int field, int ColNumb)
{
//  TCHAR TempString[256],FileString[2048];
    PBYTE pbData;


    HRESULT hr = pSample->GetPointer(&pbData);
    if (FAILED(hr)) {
      return hr;
    }

        int Y,Cb,Cr;
        int R,G,B;
        unsigned char uR,uG,uB;
//      for (int row = SAMPLE_HEIGHT/2-1; row >= 0 ; row--){
        for (int row = 0; row < SAMPLE_HEIGHT/2; row++){
                Y =  ((int)(pbData[(2*row + (long)(field))*SAMPLE_WIDTH*2 + ColNumb*2])) - 16;
                Cb = ((int)(pbData[(2*row + (long)(field))*SAMPLE_WIDTH*2 + ColNumb*2 -
2*(ColNumb%2) + 1])) - 128;
                Cr = ((int)(pbData[(2*row + (long)(field))*SAMPLE_WIDTH*2 + ColNumb*2 -
2*(ColNumb%2) + 3])) - 128;
                R = (int)(255.0*(0.00456621 * (float)Y + 0.00625893 * (float)Cr));
                G = (int)(255.0*(0.00456621 * (float)Y - 0.00153632 * (float)Cb - 0.00318811 *
(float)Cr));
                B = (int)(255.0*(0.00456621 * (float)Y + 0.00791071 * (float)Cb));
//              rgb(:,:,1) = .00456621 * in(:,:,1) + .00625893 * in(:,:,3);
//              rgb(:,:,2) = .00456621 * in(:,:,1) - .00153632 * in(:,:,2) - .00318811 * in(:,:,3);
//              rgb(:,:,3) = .00456621 * in(:,:,1) + .00791071 * in(:,:,2);
                uR = (R > 255 ? 255:(R < 0 ? 0 : R));
                uG = (G > 255 ? 255:(G < 0 ? 0 : G));
                uB = (B > 255 ? 255:(B < 0 ? 0 : B));

                m_pDump->Write((PBYTE)&uR, (long)1);
                m_pDump->Write((PBYTE)&uG, (long)1);
                m_pDump->Write((PBYTE)&uB, (long)1);
```

```
        }
        return NOERROR;
}
//
// EndOfStream
//
```

Summary of figures



Summary of figures (cont.)



Data-base creation

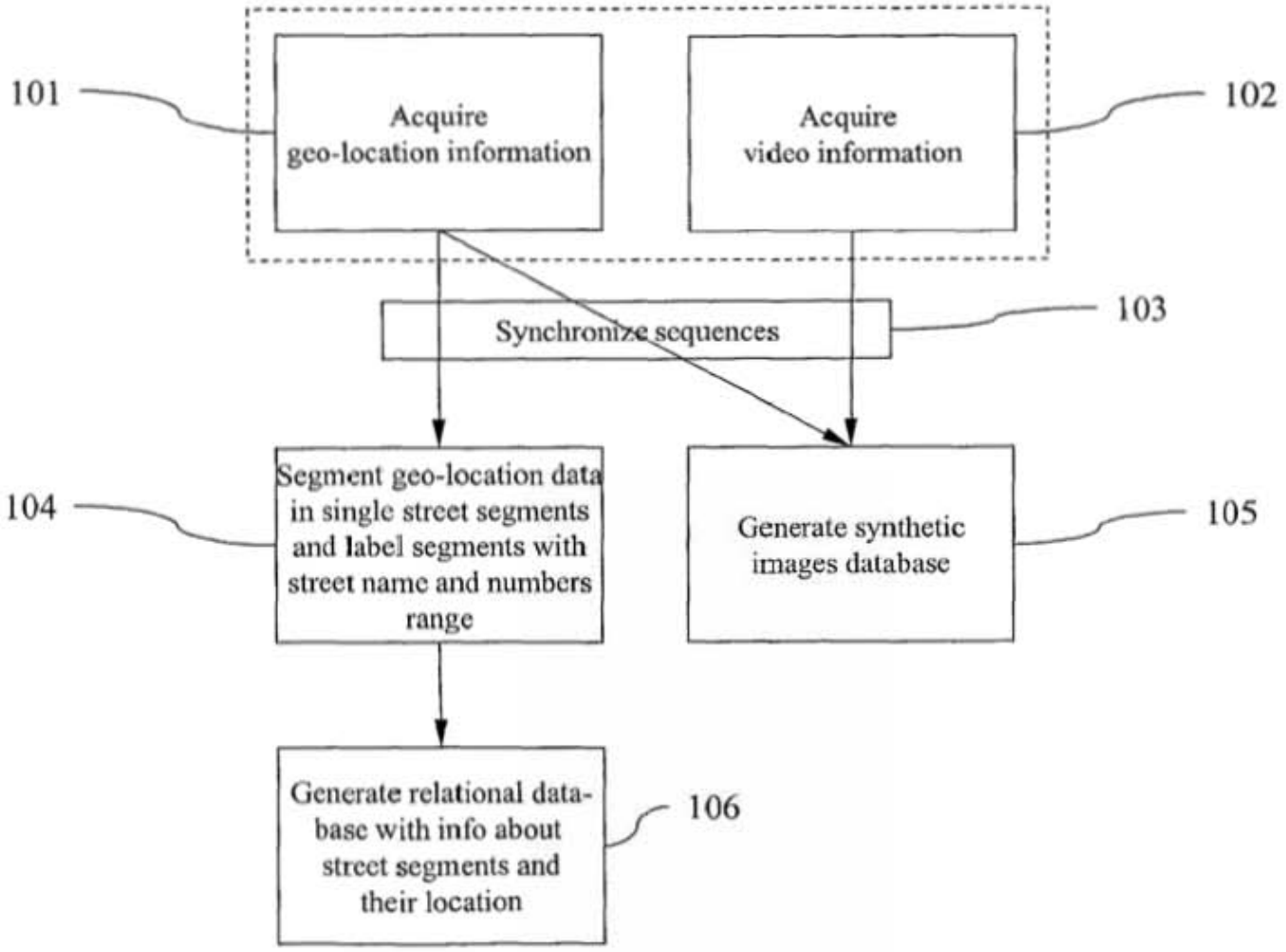


Fig. 1

Acquire video and geo-location information

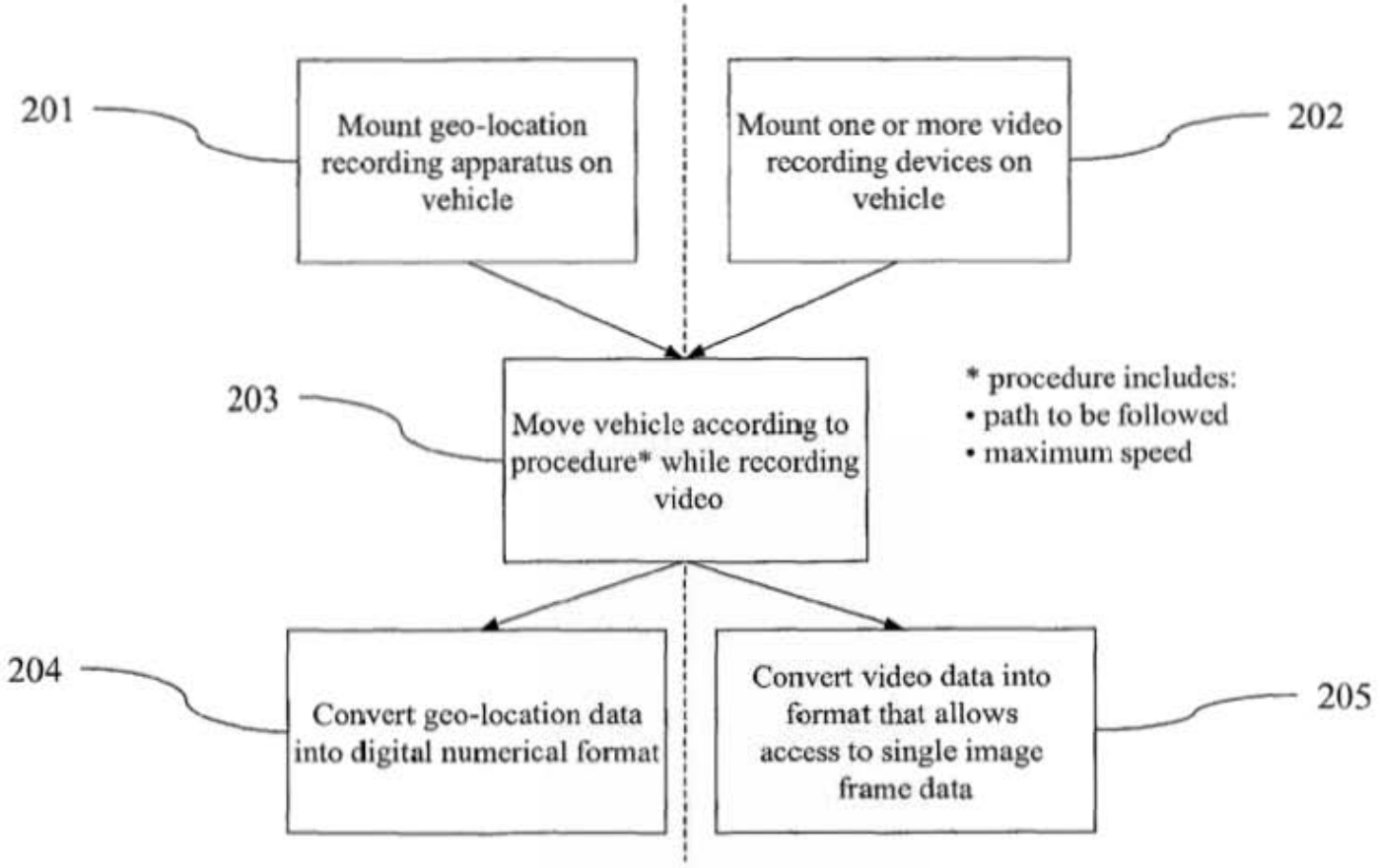


Fig.2

Synchronize sequences (1)

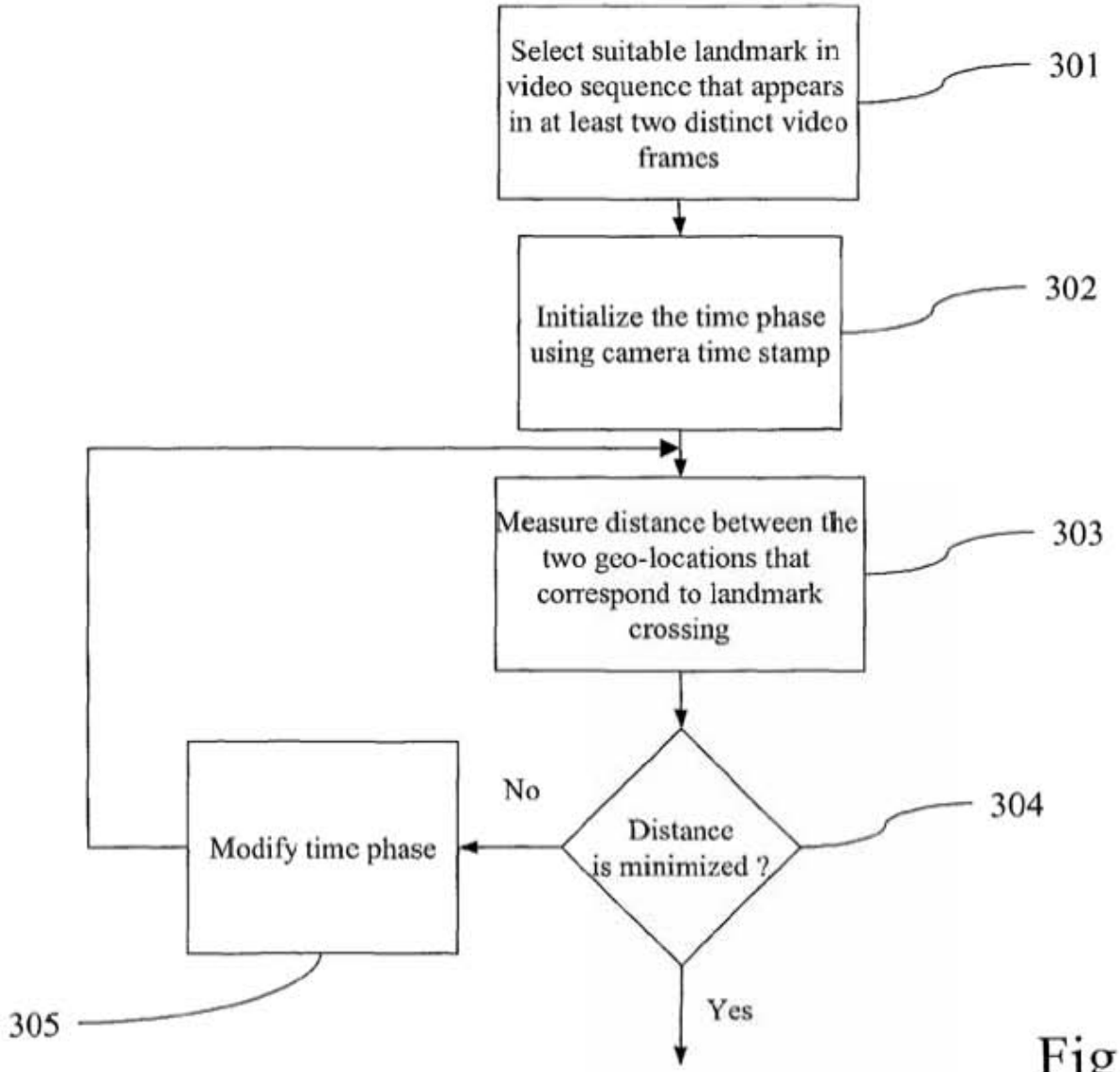


Fig.3

Synchronize sequences (2)

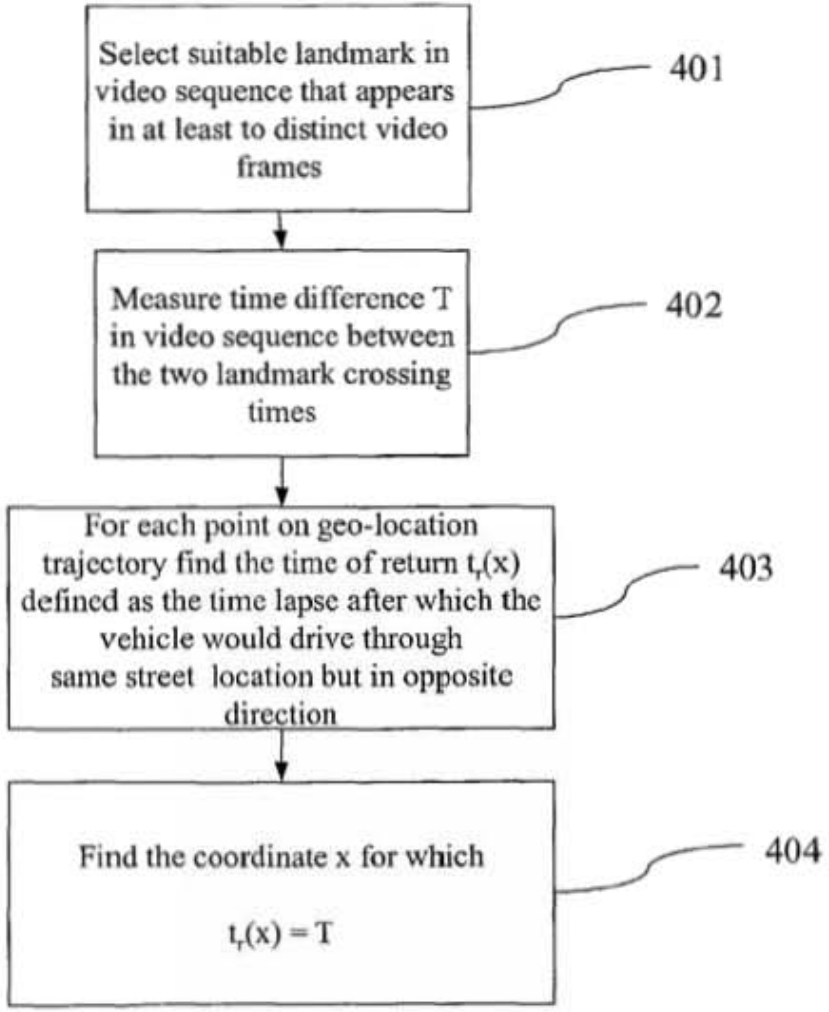


Fig.4

Synchronize sequences (3)

Hardware solution:

Time stamp video frame with UTC time in real-time using a UTC clock generator

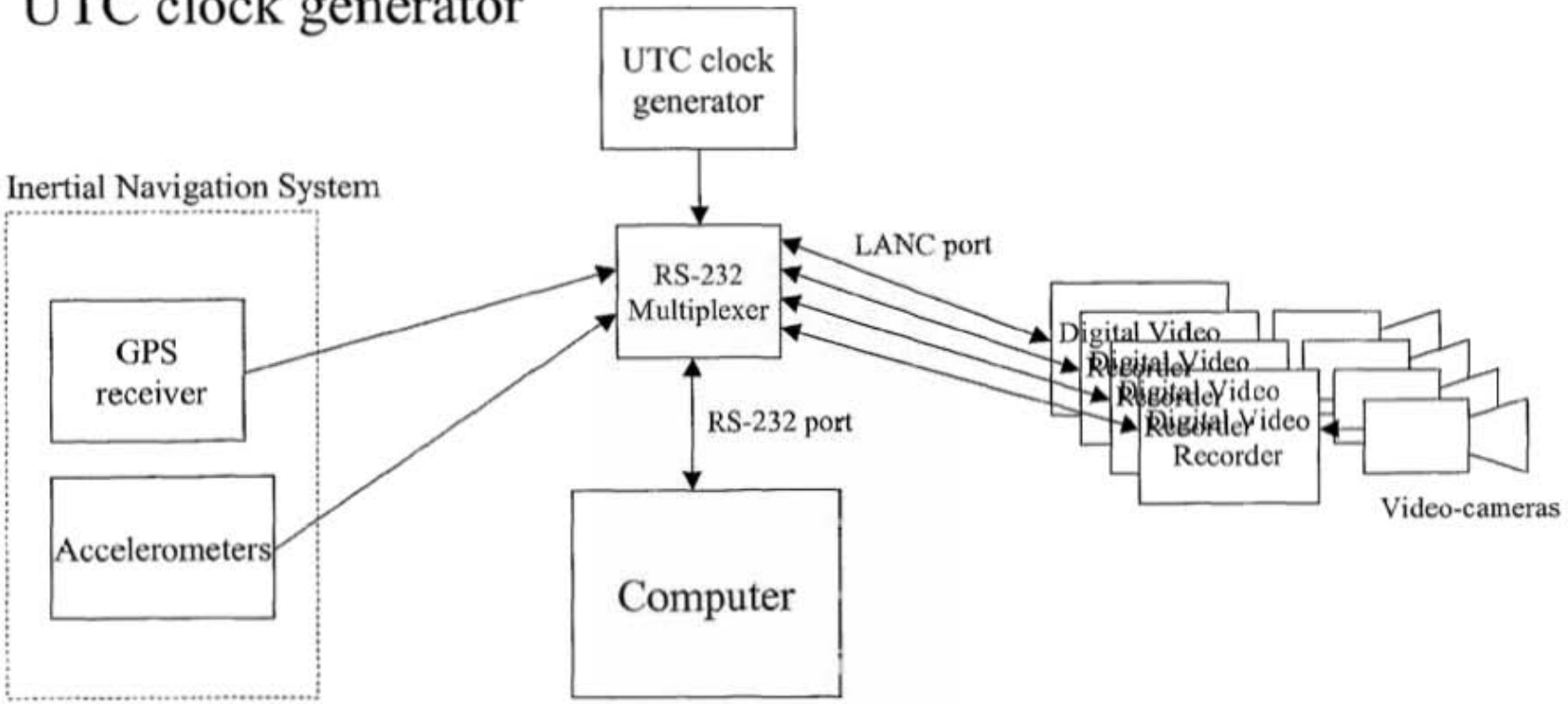


Fig.5

Synchronize sequences (4)

Using optical flow to measure vehicle speed and than register this

signal with the speed signal computed from the geo-location coordinates

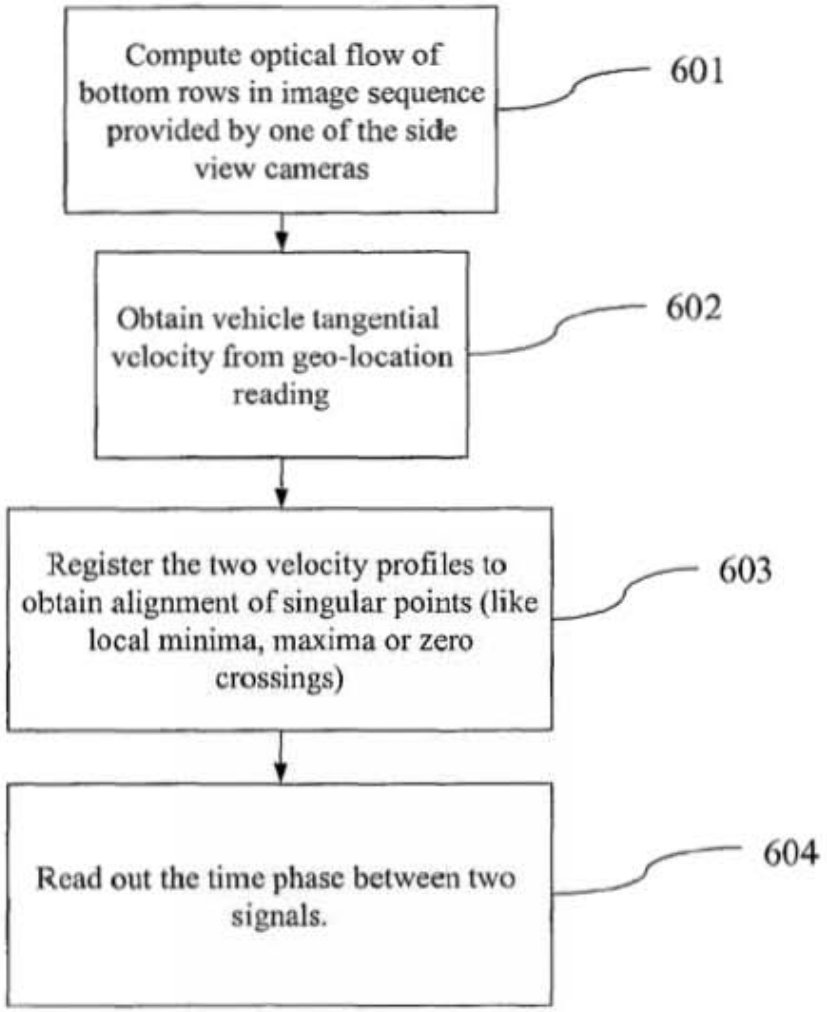


Fig.6

Segment and label geo-location data

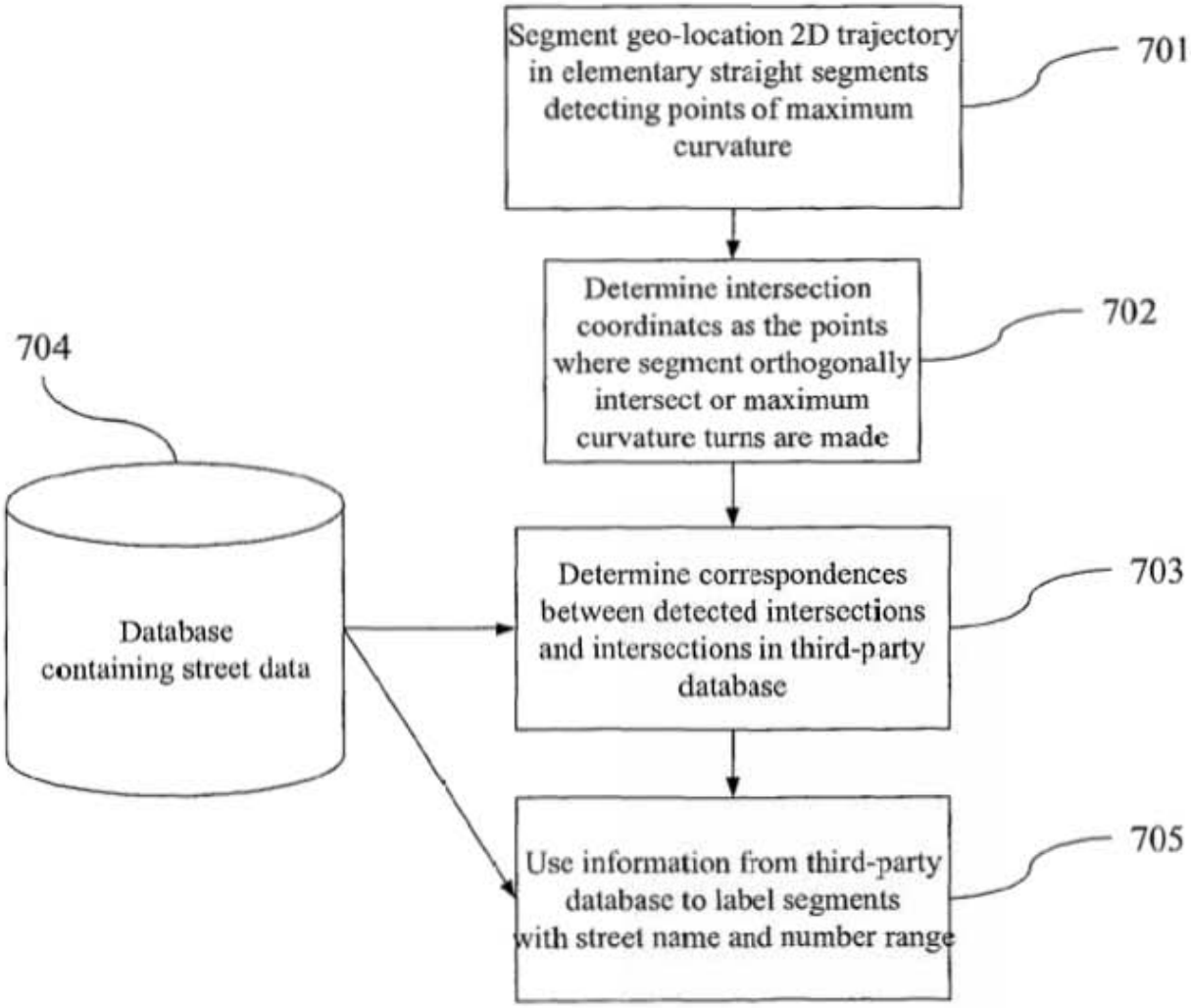


Fig.7

Generate synthetic images database

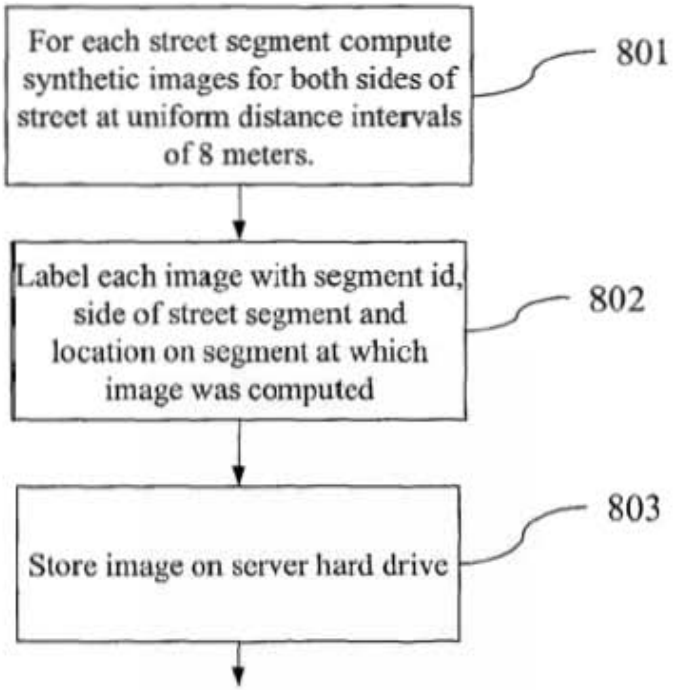


Fig.8

Compute synthetic images

Parameters:
- L - length in meters covered by synthetic view
- N - number of columns in synthetic panorama

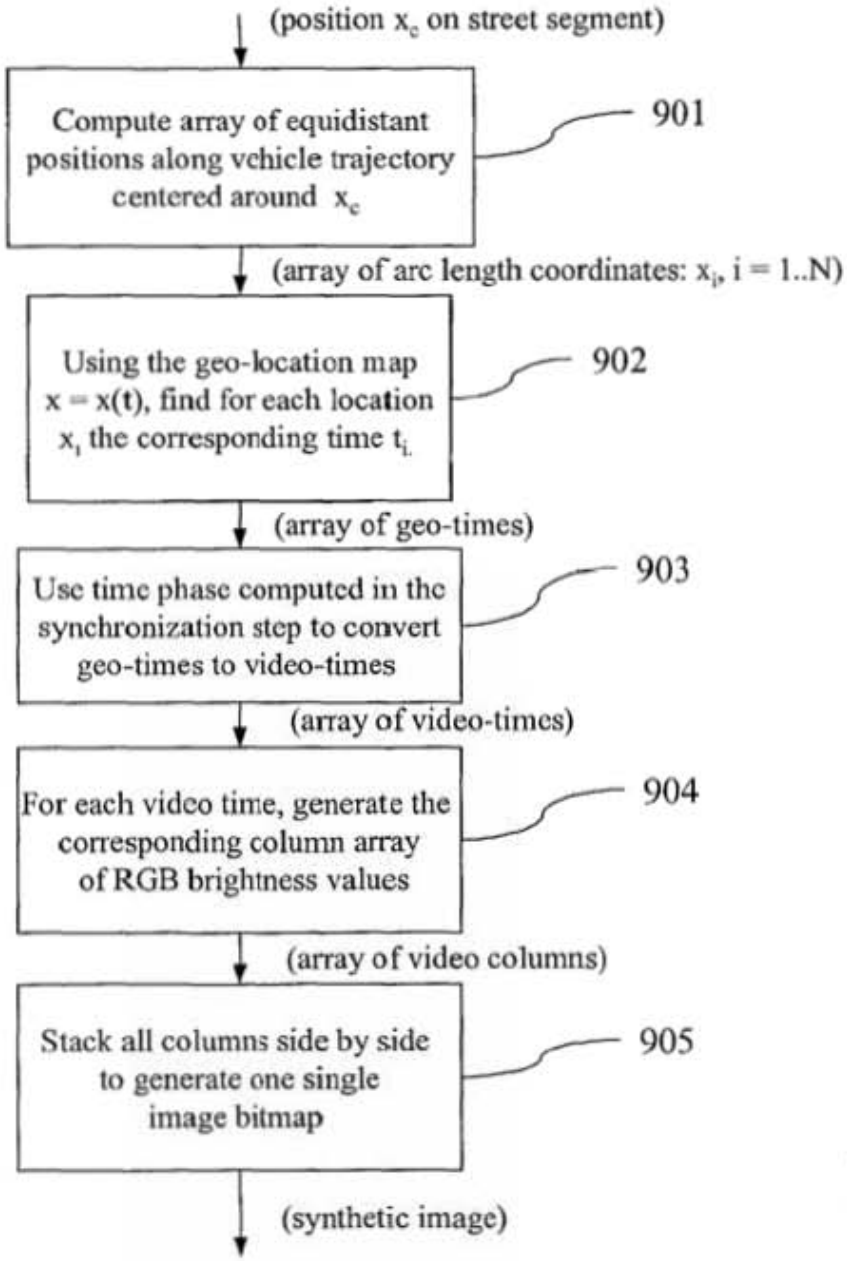


Fig.9

Generate column of synthetic image (1)

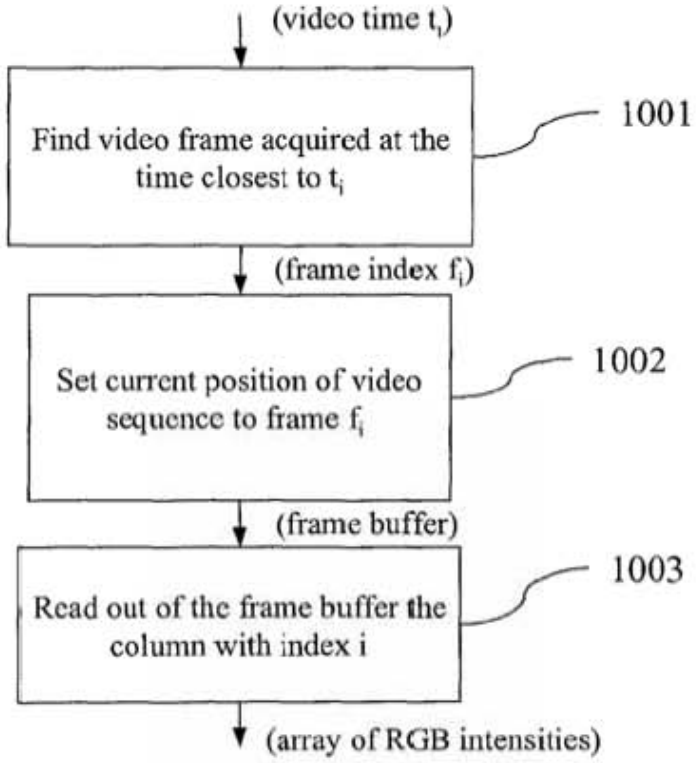


Fig.10

Generate column of synthetic image (2)

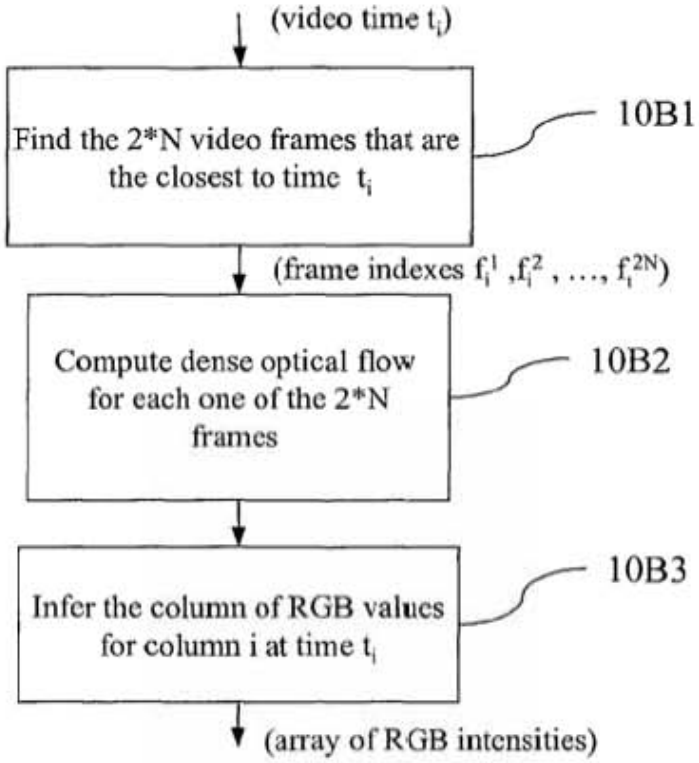
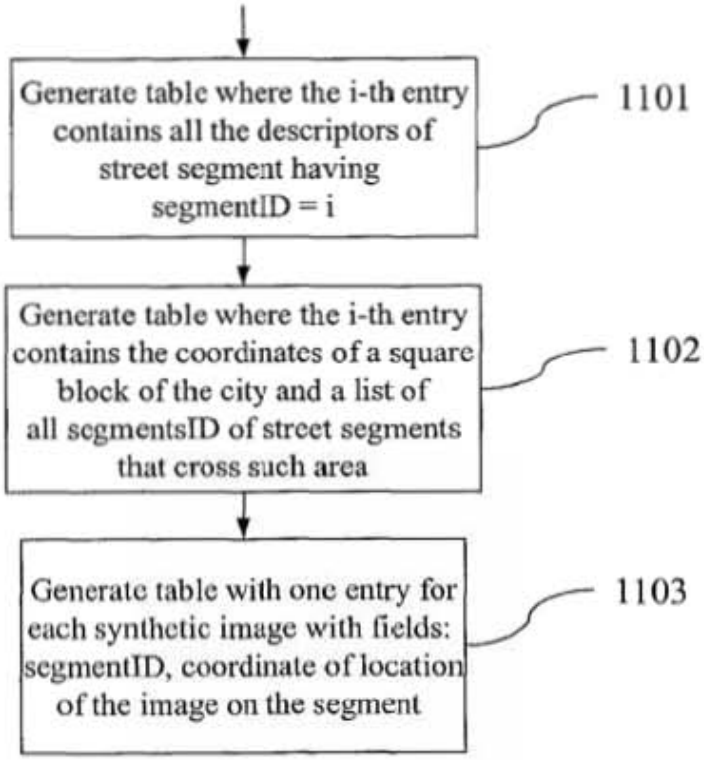


Fig.10B

Generate relational data-base



Example of layout of tool for visual exploration of geographic areas

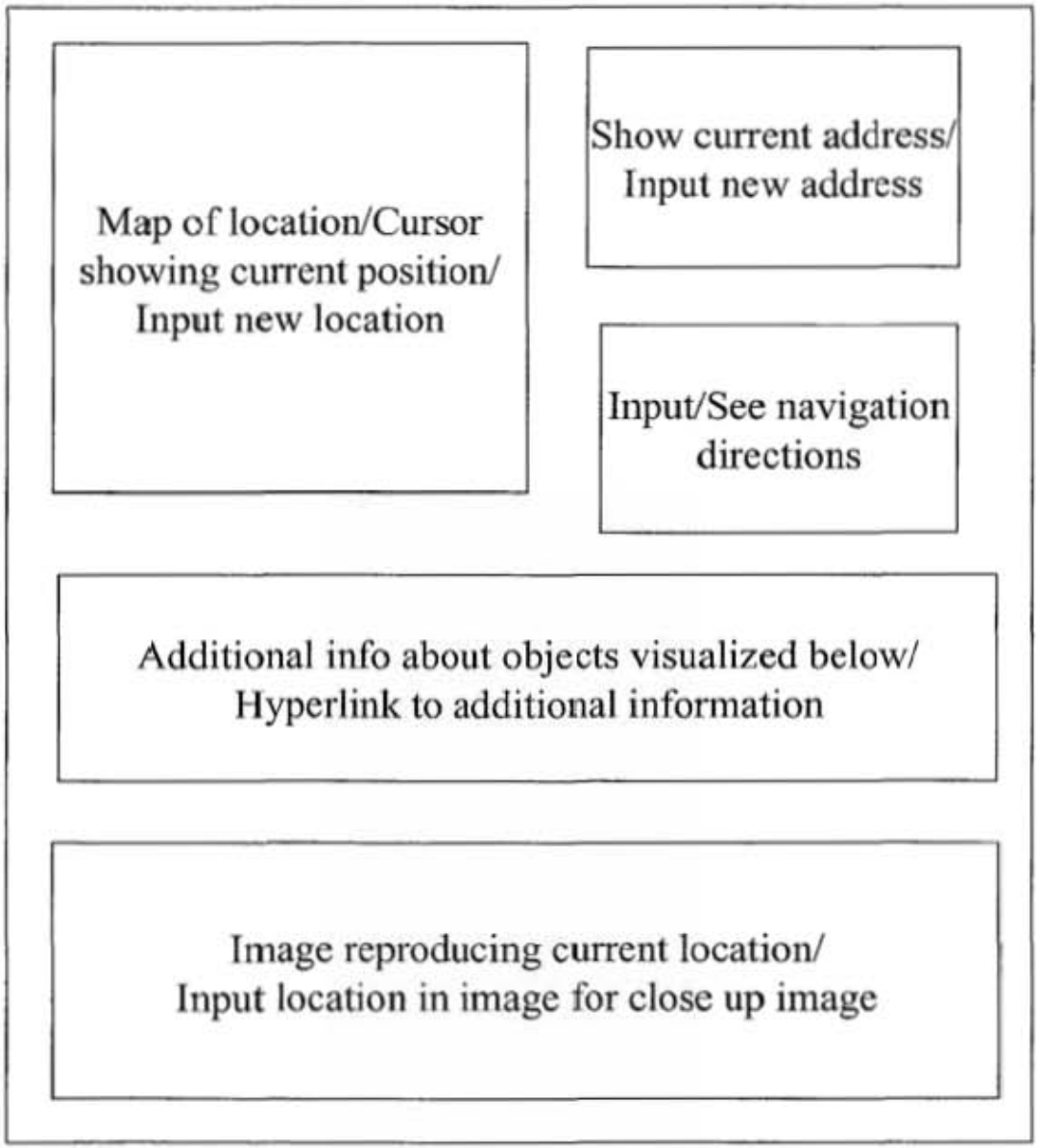


Fig.12

Implementation of address lookup

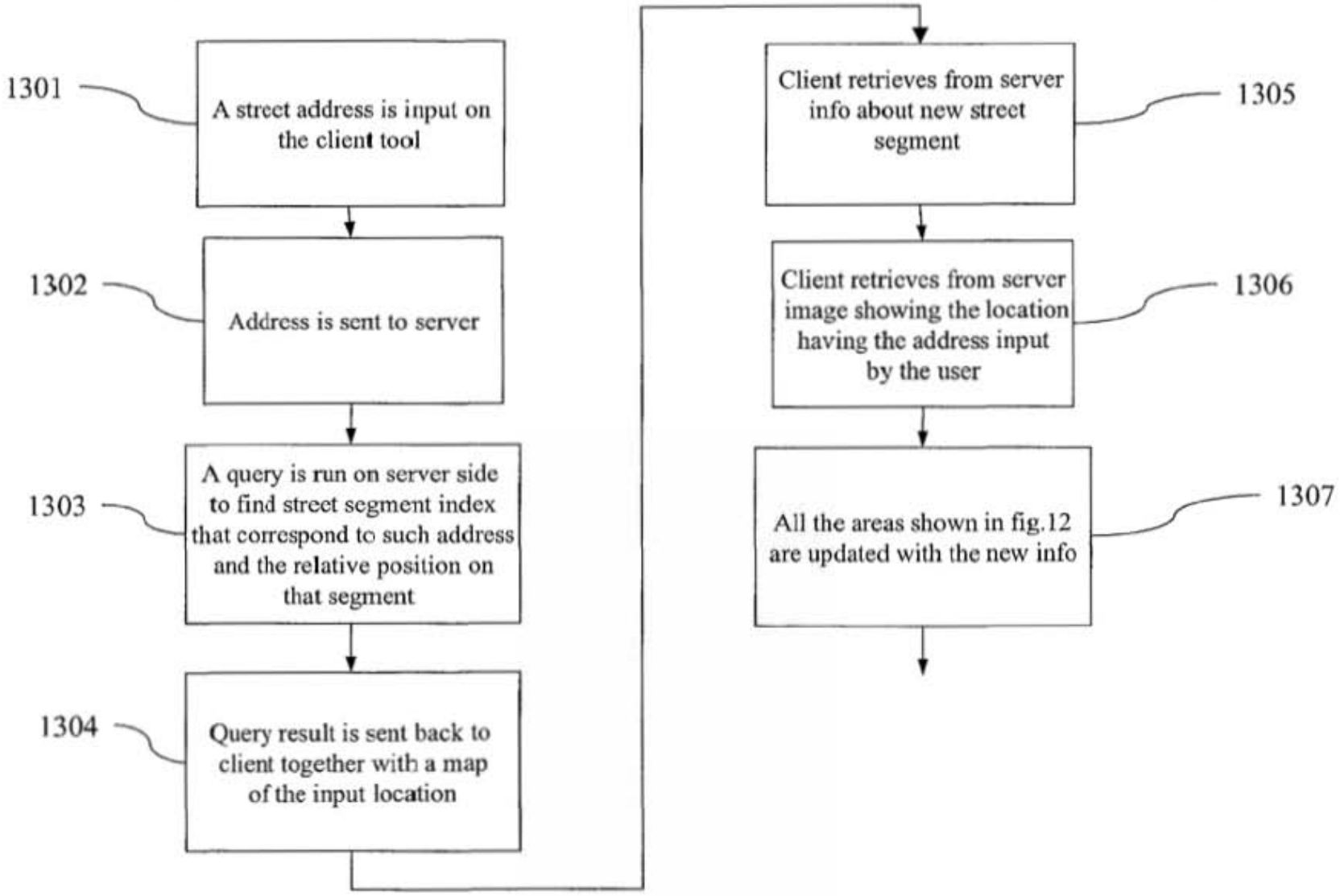


Fig.13

Implementation of input location on map

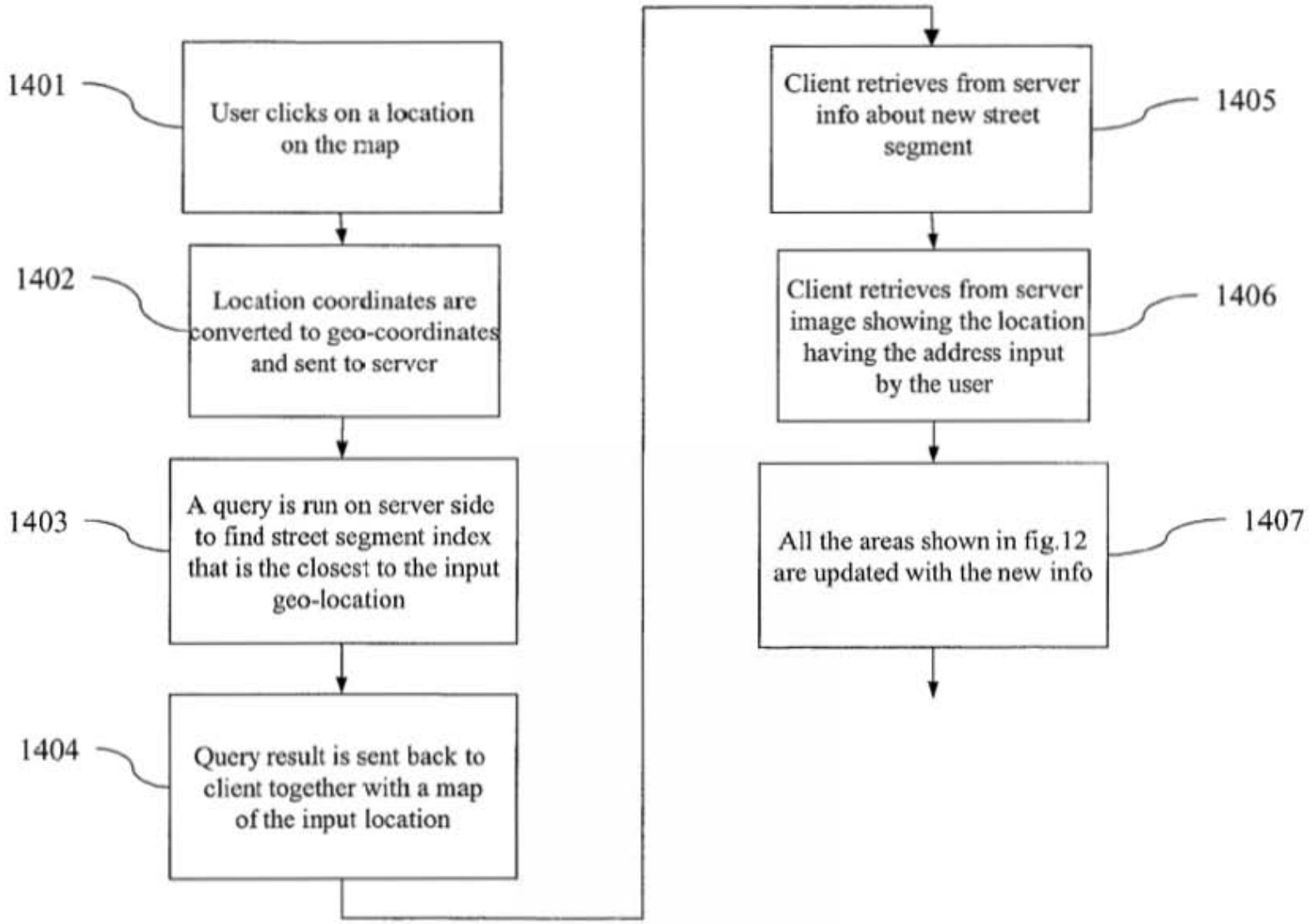


Fig.14

Implementation of input navigation directions

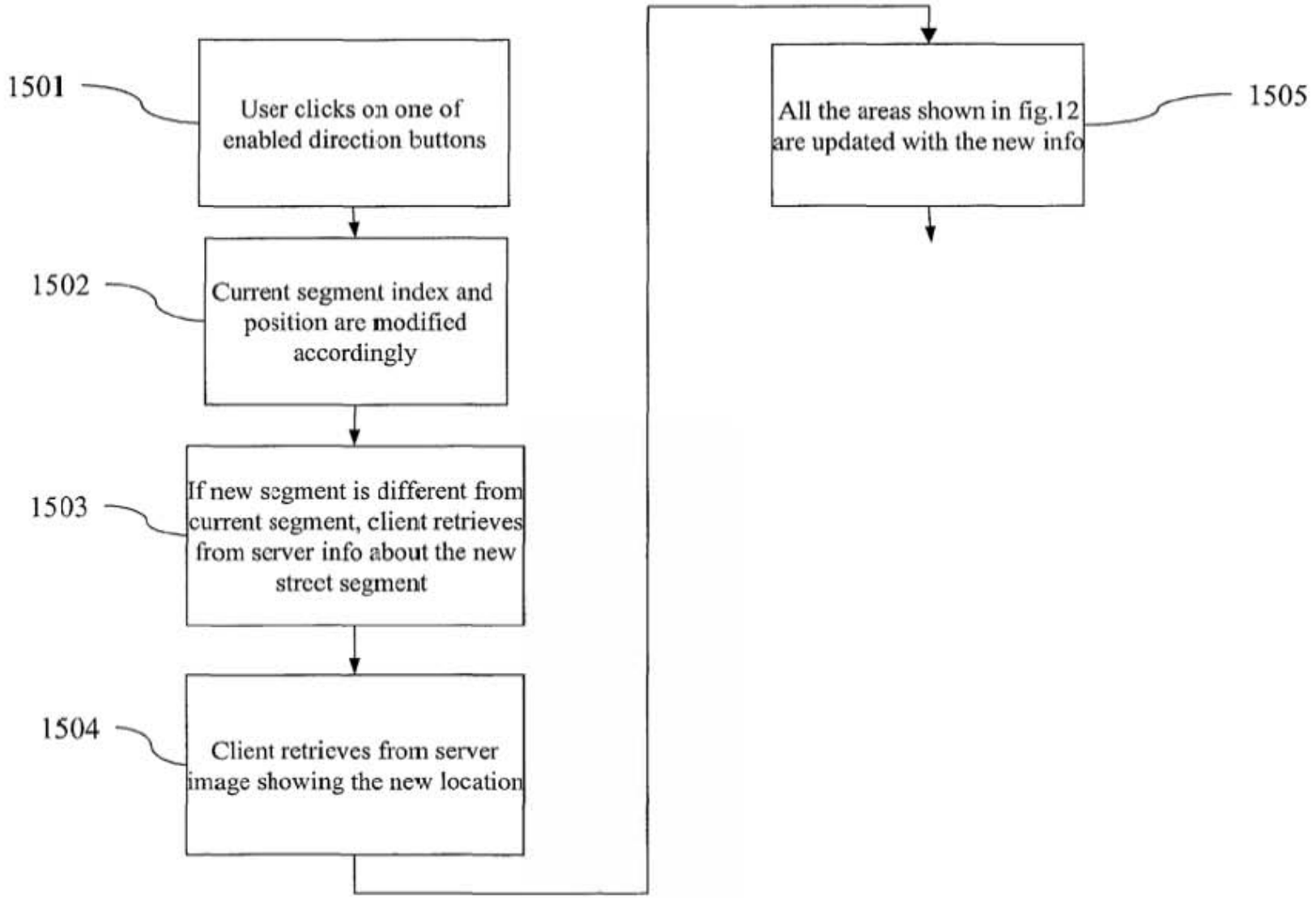


Fig.15

Implementation of selection of close-up image

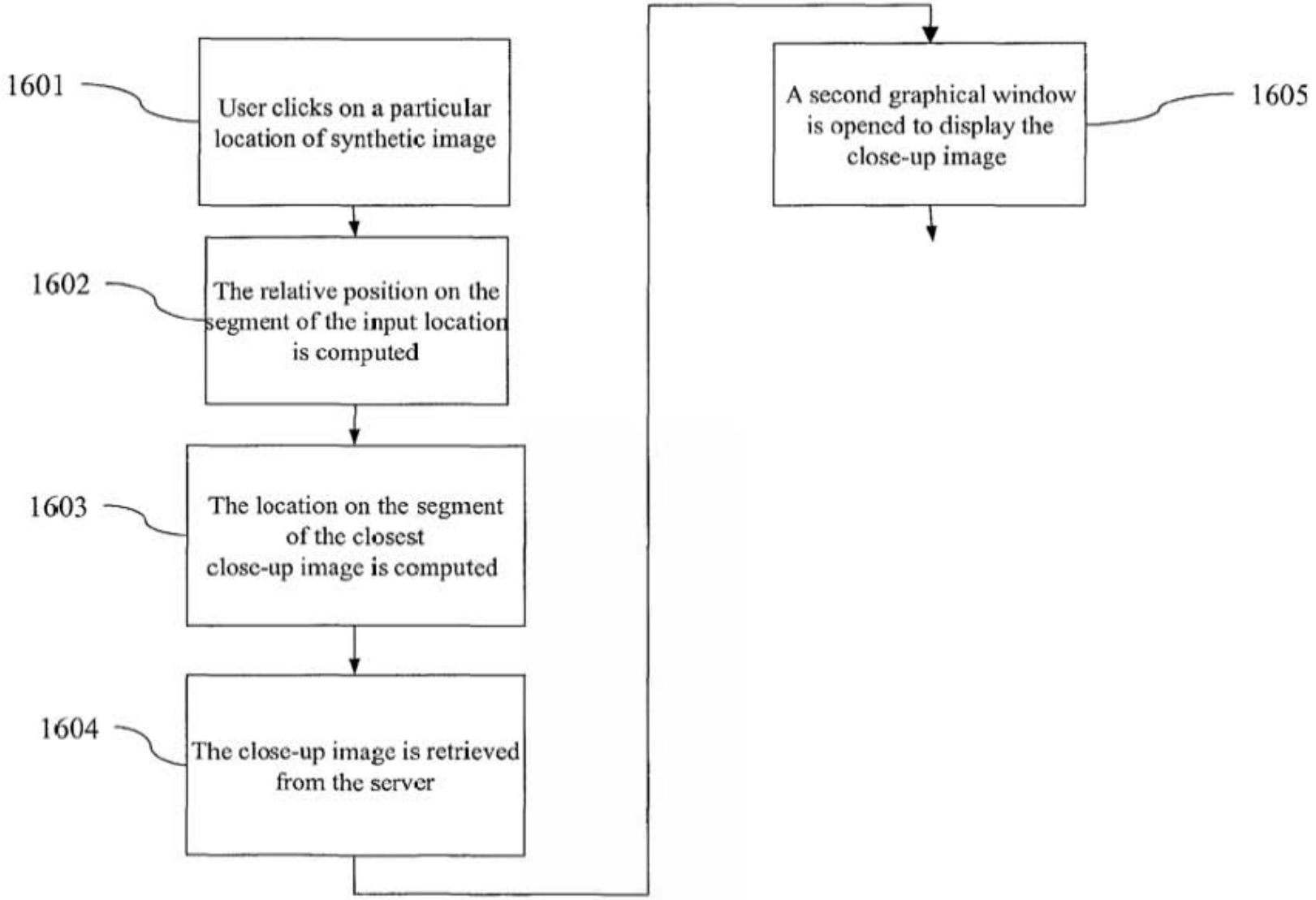


Fig.16

Implementation of visualization of info about businesses shown in synthetic image

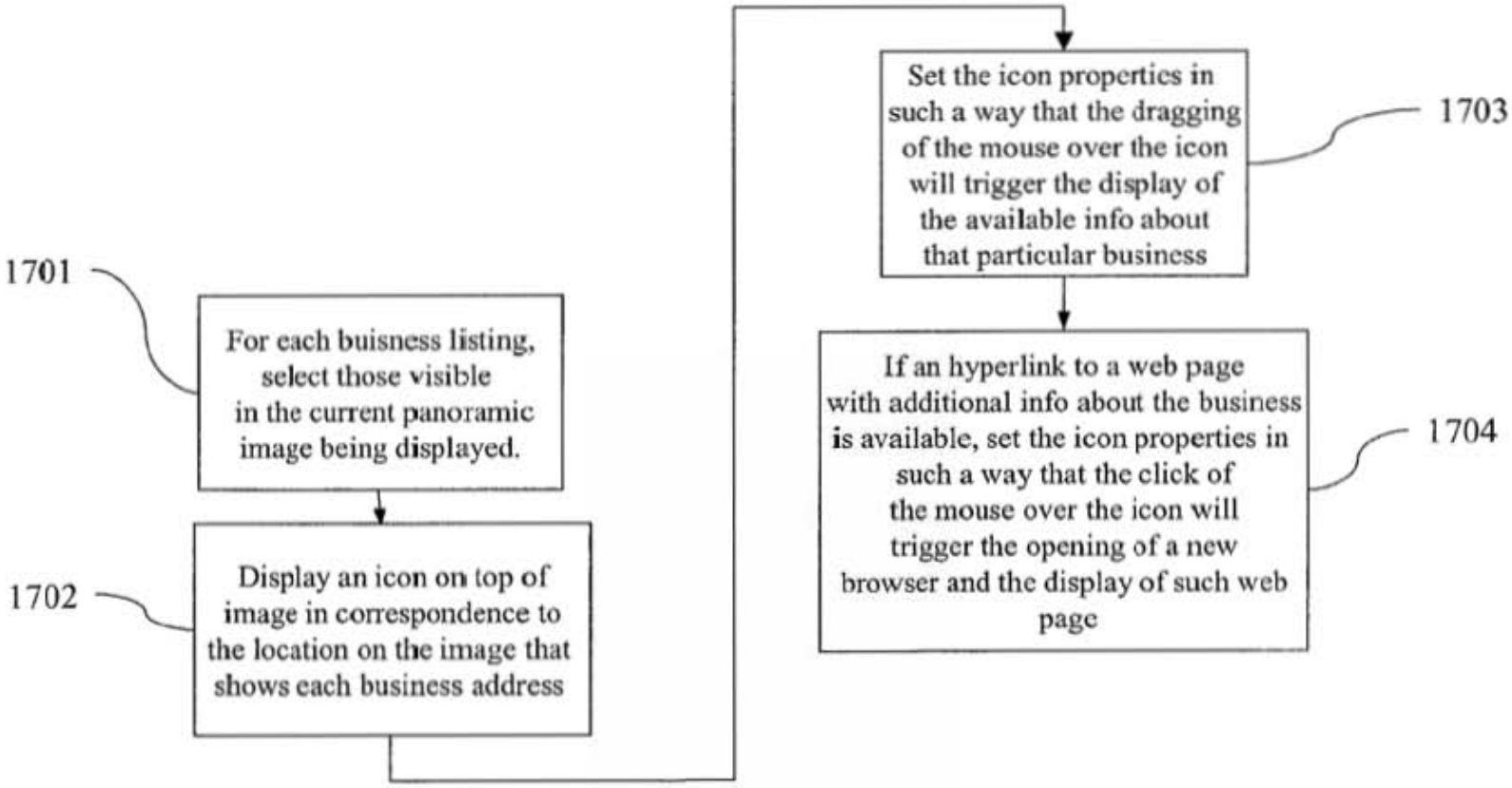


Fig.17

Info stored on client side

The array of descriptors for the current street segment includes:

- Street name
- Side of hub (one of North, South, East, or West)
- Street segment id
- End-point coordinates of map segment corresponding to current segment
- Array of segments adjacent to segment origin
- Array of segments adjacent to segment end
- Street number offset (even, odd)
- Array of coordinates of available panoramic views (even, odd)
- Array of records containing info about businesses belonging to street segment