14658

Traffic Monitoring Using Satellite and Ground Data: Preparation for Feasibility Tests and an Operational System

Mark R. McCord[1,2]
Prem K. Goel[1,3]
Carolyn J. Merry[1,2]

[1]Center for Mapping
[2]Department of Civil and Environmental Engineering and Geodetic Science
[3]Department of Statistics

The Ohio State University
Columbus, Ohio

Prepared in cooperation with the
Ohio Department of Transportation and the
U.S. Department of Transportation, Federal Highway Administration

ODOT Agreement No. 8494, State Job No. 14658(0)

Final Report

Research Foundation Project 864236/733062
Columbus, Ohio

April, 2000

:laimer: The contents of this report reflect the views of the authors who are responsible for ıcts and the accuracy of the data presented herein. The contents do no necessarily reflect the ficial views or policies of the Ohio Department of Transportation or the Federal Highway Administration. The report does not constitute a standard, specification or regulation.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
FHWA/OH-2000/006		3 1980 00017 1401

4. Title and Subtitle	5. Report Date
TRAFFIC MONITORING USING SATELLITE AND GROUND DATA: PREPARATION FOR FEASIBILITY TESTS AND AN OPERATIONAL SYSTEM	April, 2000
	6. Performing Organization Code
	8. Performing Organization Report No.
7. Author(s) Mark R. McCord, Carolyn J. Merry	
	10. Work Unit No. (TRAIS)
9. Performing Organization Name and Address The Ohio State University Department of Civil Engineering Columbus, OH 43210	11. Contract or Grant No. State Job No. 14658(0)
	13. Type of Report and Period Covered Final Report
12. Sponsoring Agency Name and Address Ohio Department of Transportation 1600 West Broad Street Columbus, OH 43223	14. Sponsoring Agency Code

15. Supplementary Notes

Prepared in cooperation with the U.S. Department of Transportation, Federal Highway Administration

16. Abstract We address three issues in this report: demonstrate that vehicles can be identified and classified accurately from satellite imagery, develop efficient image processing methods, and determine methods to integrate the imagery with ground-based data and assess the value of this integration. Field tests were replicated, where aerial photographs were used to simulate satellite imagery. We developed software to automate many of the calculations involved, and the empirical results show that our approach and software work well. We notice discrepancies between image- and ground-based data that lead us to propose that there are inevitable differences between image- and ground-based data sets that cannot be attributed to misidentification of vehicles in the images. Automated image processing must be used to perform the detection and classification of vehicles. A more robust methodology that first identifies dynamic (moving) pixels by subtracting an image of a highway segment under *current* conditions from a steady-state *background* image intended to represent the same segment with no vehicles present is described. The effects of different lighting conditions in the current and background images are reduced by first transforming grey tones of one of the images. We develop an iterative, maximum likelihood-based procedure that requires an *a prior* estimate of the probability that a random pixel in the current image is dynamic. Tests on images generated from computer simulations and on images obtained from scanned aerial photographs show promise. The limited temporal coverage of polar-orbiting satellites has led us to focus on using satellite imagery to improve estimates of Average Annual Daily Traffic (AADT) on highway segments and Vehicle Miles Traveled (VMT) over the network of these segments. We develop methods to simulate the improvements in AADT and VMT estimates produced by combining data obtained on time scales consistent with satellite orbits with data collected on the ground. Numerical results indicate the potential of satellite-based data to complement ground-based data and markedly reduce the errors in AADT or VMT estimation and the personnel required to obtain sufficient ground data to produce a given level of accuracy.

17. Key Words	18. Distribution Statement
remote sensing, image processing, mapping traffic data collection, automatic traffic recorders	No Restrictions. This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified		

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

Traffic Monitoring Using Satellite and Ground Data: Preparation for Feasibility Tests and an Operational System

Mark R. McCord[1,2]
Prem K. Goel[1,3]
Carolyn J. Merry[1,2]

[1]Center for Mapping
[2]Department of Civil and Environmental Engineering and Geodetic Science
[3]Department of Statistics

The Ohio State University
Columbus, Ohio

Prepared in cooperation with the
Ohio Department of Transportation and the
U.S. Department of Transportation, Federal Highway Administration

ODOT Agreement No. 8494, State Job No. 14658(0)

Final Report

Research Foundation Project 864236/733062
Columbus, Ohio

April, 2000

Disclaimer: The contents of this report reflect the views of the authors who are responsible for the facts and the accuracy of the data presented herein. The contents do no necessarily reflect the official views or policies of the Ohio Department of Transportation or the Federal Highway Administration. The report does not constitute a standard, specification or regulation.

TABLE OF CONTENTS

APPENDICES

Appendix *Page*

ILLUSTRATIONS

Figure *Page*

TABLES

Table *Page*

Executive Summary

Satellite imagery could conceivably be added to data traditionally collected in traffic monitoring programs to allow wide spatial coverage unobtainable from ground-based sensors in a safe, off-the-road environment. Previously, we estimated that 1-m resolution panchromatic imagery should allow accurate vehicle counts and rough vehicle classifications, while large vehicles might be accurately detected with only 4-m resolution. At least three private groups are planning to market such high-resolution satellite data in the near future, but several issues must be addressed before these data could be used to complement traffic monitoring programs. This report addresses the following issues:

- demonstrating that vehicles can be identified and classified accurately from satellite imagery;
- developing efficient image processing methods; and
- determining methods to integrate the imagery with ground-based data and assessing the value of this integration.

Previously, we designed a process to compare image data with data obtained from ground-based sensors to investigate the accuracy in identifying and classifying vehicles from imagery. We also tested the process using aerial photographs to simulate satellite imagery. In our new work, we replicate these field tests and develop software that automates many of the analytical components involved with these tests. The software could eventually be used in tests conducted with real satellite data. The empirical results of our new field tests show that our approach and software work well. However, we notice discrepancies between image- and ground-based data that lead us to propose that there are inevitable differences between image- and ground-based data sets that cannot be attributed to misidentification of vehicles in the images. Therefore, data collected from ground-based sensors should not be considered as absolute ground truth against which image-based data should be evaluated. Further work is warranted to reduce the magnitude of these inevitable differences and to determine how to work with such differences when determining the accuracy of vehicle identification and classification from image-based data. Additional consideration should also be given to operational differences in the tests we have conducted using simulated satellite imagery (scanned aerial photographs) and the ultimate tests of interest – those using real satellite data. For example, consideration should be given to anticipated data formats and the ease with which highway segments of interest can be identified and delimited in the images.

Based on our experience with simulated high-resolution imagery, we are optimistic that an individual could visually detect and develop vehicle classifications from 1-m satellite imagery. However, to be useful in practice, automated image processing must be used to perform the detection and classification. We had previously developed rules that could be coupled with thresholding methods to count and classify vehicles using panchromatic imagery. This approach worked well under conditions where vehicle shadows were pronounced, but it did not perform well under different lighting conditions. We are now developing a more robust methodology that first identifies dynamic (moving) pixels by subtracting an image of a highway segment under *current* conditions from a steady-state *background* image intended to represent the same segment with no vehicles present. The effects of different lighting conditions in the current and background images are reduced by first transforming grey tones of one of the images. We develop an iterative, maximum likelihood-based procedure that requires an *a prior* estimate of the probability that a random pixel in the current image is dynamic. Tests on images generated from computer simulations and on images obtained from scanned aerial photographs show the promise of this approach and its robustness to the prior probability estimates required. Future work is needed to refine the image processing components we have been developing, to test them further, and to incorporate them with vehicle classification modules that would operate on the set of dynamic pixels identified.

The limited temporal coverage that would be possible from a sensor carried on a satellite in a nongeostationary orbit has led us to focus on using satellite imagery to improve estimates of Average Annual Daily Traffic (AADT) on highway segments and Vehicle Miles Traveled (VMT) over the network of these segments. We develop methods to simulate the improvements in AADT and VMT estimates produced by combining data obtained on time scales consistent with satellite orbits with data collected on the ground. Numerical results indicate the potential of satellite-based data to complement ground-based data and markedly reduce the errors in AADT or VMT estimation and the personnel required to obtain sufficient ground data to produce a given level of accuracy. These encouraging results were obtained when using methods similar to those traditionally employed in practice. We improve estimates further by developing a method designed to take advantage of the assumed data models. However, we expected to see greater improvements when using this method. We, therefore, feel that this method can be refined and that other methods can be developed to exploit the different spatial-temporal natures of the satellite- and ground-based data.

Section 1. Introduction

This report documents our continued research into the feasibility of using data obtained from satellite images to improve estimates of interest in traffic monitoring programs. Using satellite imagery is attractive for traffic monitoring programs, since imagery would allow wide spatial coverage unobtainable from ground-based sensors. In addition, sensors onboard satellites are *off-the-road*, and, therefore, there is no disruption to traffic flow or increased hazard to personnel during installation and repair. Moreover, high-resolution satellite imagery will soon be available for the first time in the non-military world.

Previously, we estimated that approximately 1-m resolution panchromatic imagery should allow accurate vehicle counts and rough vehicle classifications, while large vehicles might be accurately detected with only 4-m resolution (McCord *et al.* 1995a, 1995b). At least three private groups are planning to market high-resolution satellite data in the near future (American Society of Photogrammetry and Remote Sensing, 1996). EarthWatch, Inc. lost the EarlyBird satellite (3-m panchromatic data) shortly after launch in December 1997. However, the company is focused on the QuickBird-1 Satellite with a 1-m panchromatic (0.45-0.9 μm) sensor and a 4-m multispectral (MS) sensor onboard. Orbital Sciences Corporation is presently developing OrbView-3, which will have 1-m panchromatic and 4-m MS sensors. In April 1999, Space Imaging EOSAT lost the Ikonos-1 satellite that was to carry 1-m panchromatic and 4-m MS sensors. Ikonos-2, an identical twin to Ikonos-1, was launched on 24 September, 1999. After an initial four-month calibration period, Ikonos images are now available for purchase by the public.

Several issues would need to be addressed before such high-resolution satellite imagery could operationally be used to complement traffic monitoring programs. This report addresses the following issues:

- To gain acceptance, it would be necessary *to demonstrate that vehicles can* indeed *be identified and classified* accurately from real satellite imagery.
- To be used operationally, it would be necessary *to develop methods that* efficiently *process image data* into data that can be used to improve traffic parameter estimation.
- To stimulate investment in implementation, it would be necessary *to assess the value* that the processed imagery data would add to traditional traffic parameter estimation *and to develop methods for integrating the data* with ground-based data to increase the value of the combined data.

Showing that the numbers of classified vehicles observed in satellite images match those obtained from ground truth data would demonstrate that vehicles could be counted and classified from satellite imagery. However, determining ground truth data comparable to the type of data observed in a satellite image would not be straightforward. To obtain the 1-m ground resolution we are seeking to detect vehicles, the sensor would need to orbit at altitudes much less than those permitting geostationary orbits, orbits where the satellite can continually image a fixed location on the earth (McCord *et al.* 1995a). The nongeostationary orbits imply that the image-based data would consist of *snapshots* of different vehicles over wide spatial areas taken at instants in time (Merry *et al.* 1996, McCord *et al.* 1995a). On the other hand, data obtained from ground sensors would consist of vehicles passing a point in space over an interval of time. Previously, we designed and field tested a process to compare the image data with data obtained from ground sensors (Merry *et al.* 1996). We used aerial photographs to simulate the satellite imagery because of the unavailability of high-resolution satellite imagery.

In Section 2, we report on new field tests, where we again used aerial photographs to simulate satellite imagery. In our new work, we also developed software to automate many of the calculations involved. The empirical results show that our approach and software work well. However, we still notice differences between vehicle classifications obtained from the image- and ground-based data. We propose that some differences are unavoidable because of the different nature of the data sets. Therefore, when conducting tests with real satellite data in the future, data obtained from ground-based sensors should not be considered as absolute ground truth. Further work seems warranted to reduce the size of the differences that can occur and to obtain a feel for the maximum difference that could be tolerated before the equivalence of the number of vehicles in the image- and ground-based data would be rejected with confidence.

Based on our experience with aerial photographs scanned to simulate 1-m imagery, we are optimistic about the ability to detect and classify vehicles from high-resolution satellite imagery. Specifically, we have always been able to visually detect in the 1-m images vehicles that appeared in the original aerial photographs. However, if such imagery is to be useful in practice, the detection and classification would need to be performed automatically.

In Section 3, we report on our progress in developing operational image processing methods for vehicle classification. The task is different from the presently popular one of detecting vehicle presence in video images of a fixed location. In video imaging, an extremely fine-resolution background of the location can be built up from thousands of frames under almost constant lighting conditions. Satellite-based images, on the other hand, will only yield pairs of overlapping images of a location, with each image in the pair taken several seconds apart, and different pairs of images taken days apart. We had previously developed classification rules that we coupled with *thresholding* methods to count and assign vehicles into two classes using panchromatic imagery. The method worked well under conditions where vehicle shadows were pronounced (McCord *et al.* 1995a, 199b). However, the method did not perform as well under different lighting conditions (Merry *et al.* 1996). We, therefore, have been developing and testing a more robust methodology. We describe this methodology in Section 3.1 and report the encouraging test results in Section 3.2. We propose further work to continue developing the components of this methodology, integrating these components into an operational program, and testing the program with simulated and real satellite data.

Although a sensor carried on a satellite in a nongeostationary orbit could image the same area on different orbits, the repeat period would be on the order of days (McCord *et al.*, 1995a). We propose that such data would be most useful for complementing traffic monitoring programs that collect and estimate state- or region-wide network traffic statistics over relatively long time periods. Compared to traditional ground-based methods, satellite imagery would detect concurrent traffic conditions on an increased number of highway segments. It could also more directly determine changes in conditions along a segment of highway. Figure 1.1 shows velocities along approximately 10 km of I-70 in Central Ohio estimated from overlapping aerial photography that we have been using to simulate satellite data.

In our work reported in Section 4, we have been concentrating on the ability of satellite-based data to improve estimates of Average Annual Daily Traffic (AADT) on highway segments and Vehicle Miles Traveled (VMT) over the network of these segments. In Section 4.1 we describe the methods we developed and coded to simulate traffic patterns and true AADT and VMT statistics and estimate these measures from observations assumed to be obtained from samples of the traffic patterns. The estimation component can use either a *traditional-based* method (what has traditionally been used to estimate these measures from ground-based sensors) or a *model-based* method that uses observations more efficiently when the data can be assumed to be compatible with a specified underlying stochastic process. In Section 4.2, we report the results of numerical studies we conducted using our software. These results indicate the potential of satellite-based data to complement ground-based data and markedly reduce the errors in AADT or VMT estimation or the personnel required to maintain an accuracy level when estimating these parameters.

In Section 5 we summarize the report and expand upon future work we feel is warranted in several areas.

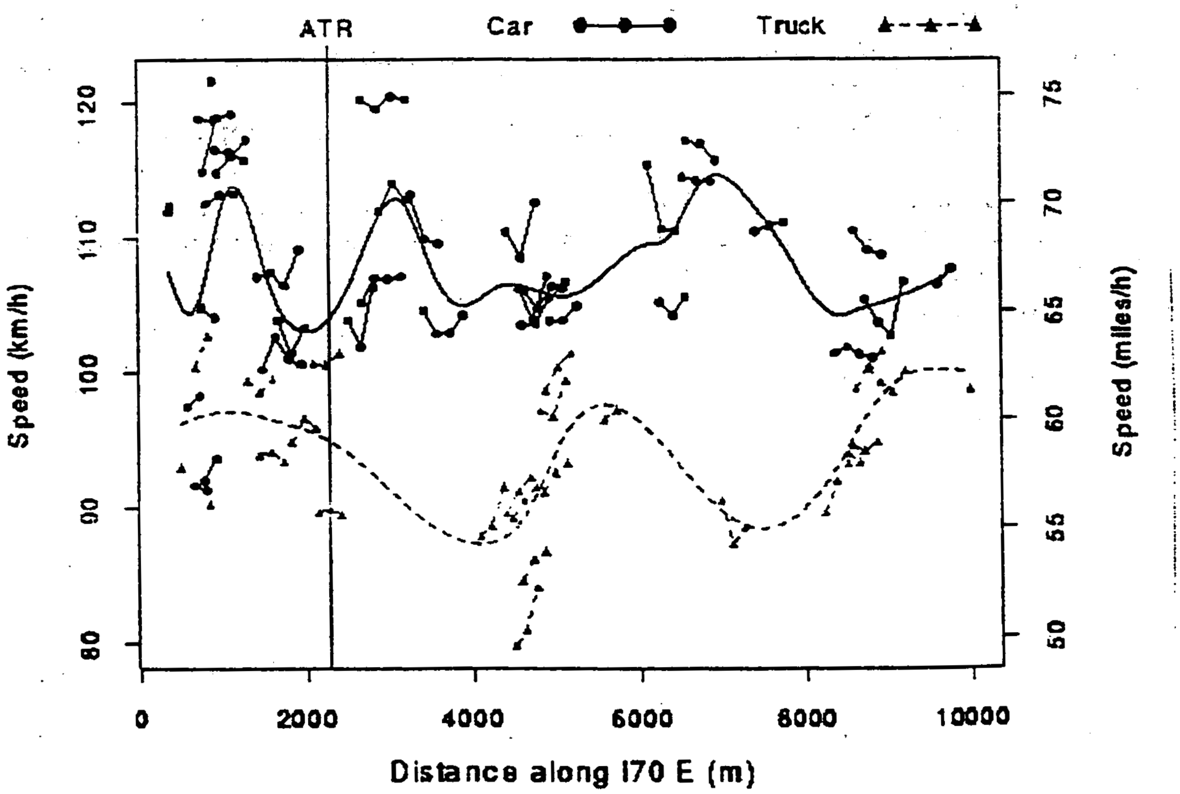


Figure 1.1. Velocity profile along I-70 E in Central Ohio, estimated from overlapping aerial photographs.

Section 2. Air-Ground Coordinated Field Tests

Our previous work (McCord *et al.* 1995a, 1995b, Merry *et al.* 1996) indicates that 1-m resolution would be sufficient to identify vehicles and distinguish between large and small vehicles in digital images scanned from panchromatic aerial photographs. It would be necessary to demonstrate that vehicles could be identified in panchromatic imagery obtained from a satellite platform to convince potential users that satellite imagery can, in reality, be used to count and classify vehicles on highway segments.

In our previous work, we compared vehicles identified in digital images scanned from aerial photographs to vehicles identified in the photographs. That is, the photographs served as the *ground truth.* When conducting tests with satellite imagery, it would be difficult to simultaneously image the dynamic highway segments with photographs and satellite imagery. Therefore, vehicle data detected from ground-based sensors would need to serve as ground truth. However, vehicle data obtained from ground-based sensors consist of vehicles passing a fixed location through time (*i.e.*, of temporal flow data at a point), whereas that collected by imagery consists of vehicles imaged at an instant across an area (*i.e.*, of spatial density data at one time). We have been developing a means to compare the ground data to that collected from the satellite. We conducted a field test similar to that previously described (Merry *et al.*, 1996) to test and refine our approach. We also wrote software that automates many of the calculations required and tested this program on the data collected. As in the previous study where we conducted the analysis manually, we scanned aerial photographs to simulate the satellite imagery.

2.1 Acquisition of Data

We conducted a new field test on 29 October 1996. The Ohio Department of Transportation's (ODOT) Bureau of Aerial Engineering obtained aerial photography of the same three highway sites in Central Ohio that were used in a test we conducted in a previous project (Merry *et al.*, 1996) – I-270 in Franklin County on the west side of Columbus, I-70 in Madison County just west of Columbus, and I-71 in Pickaway County just southwest of Columbus (see Figure 2.1). Photographs were obtained at a scale of 1 in. = 400 ft with the highway centerlines located approximately in the center of the photos. The recorded weather indicated high overcast clouds, scattered at 1800 m (6000 ft).

While the aerial photographs were being taken, ODOT's Bureau of Technical Services was collecting vehicle data passing traffic sensors embedded in the highway. For each direction of the I-70 and I-71 facilities, volume-by-length sensors were used to collect 1-minute volumes by two length classes – under 20 ft (6.1 m) and 20 ft (6.1 m) and over. For each direction of the I-270 facility, weigh-in-motion sensors were used to record FHWA vehicle class and the time to the nearest second that the vehicle passed the sensor.

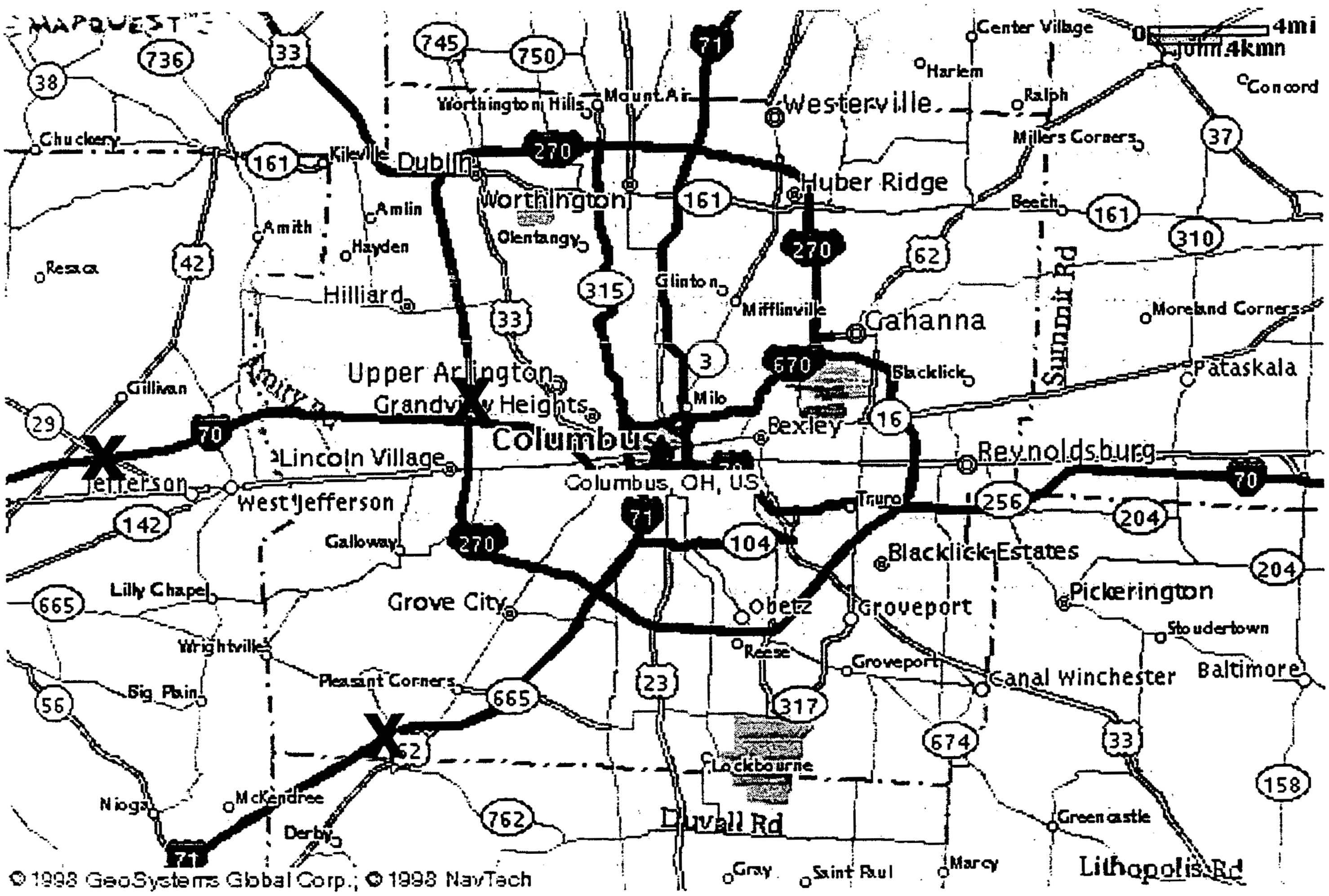


Figure 2.1. Site location map showing the I-270, I-70 and I-71 field sites used in the 1996 field test.

To provide additional control, we videotaped traffic in all but the I-71 southbound directions during the data collection period. The videotape had time stamps to the minute.

We obtained the aerial photographs and ground sensor data in the same formats as those described in Merry *et al.* (1996).

2.2 Analysis of Data

Our analysis is similar to what we developed and documented in Merry *et al.* (1996). The process compares the number of vehicles in a class passing a ground traffic sensor during a specified time interval to a projection of the number in that class that would pass the location of the sensor during the same time interval. The projections are based on vehicle locations and speeds obtained in the imagery. As such, the comparisons will be influenced not only by how well vehicles can be identified in the images, but also by how well the times that the identified vehicles will arrive at the sensor location, which we denote X^{sens}, can be predicted.

We considered two vehicle classes, small and large, which we call *cars* and *trucks,* for simplicity. We based the classes on size, since it is a parameter that could conceivably be distinguished in images. In the volume-by-length sensor data, we classed vehicles less than 20 ft (6.1 m) long as cars and vehicles 20 ft (6.1 m) or longer as trucks. In the weigh-in-motion data, we considered vehicles in FHWA classes 1, 2, 3, and 5 to be in our car category and vehicles in the other classes to be in our truck category. The sensor data is provided by lane, but we aggregated across lanes to obtain classified counts during a time interval by direction (see Merry *et al.*, 1996). In this way, the numbers of cars and trucks passing the sensor during a specified time interval were readily available from the data recorded by the ground sensor. The time intervals are those recorded by the ground sensor.

We visually classified vehicles in the aerial photographs as cars or trucks based on size. We also visually identified identical vehicles in different photographs and assigned each vehicle a 2-part identifier, where the first part indicated its class (*C* or *T*, for car or truck) and the second part (an integer number) allowed it to be identified as the same vehicle in different images: a vehicle with the same identifier in different photographs was believed to be the same vehicle.

The photographs were scanned and saved as digital 8-bit image files. The x,y locations of the vehicles were digitized from these image files. The times that the vehicles were imaged and vehicle identifiers were manually added to the file. Highway edgelines were also digitized from these image files. The images were placed in a common x,y coordinate system. This consisted of registering the images by identifying points that were common to pairs of images. The digitized locations of vehicles at specified times, the two-part identifiers of these vehicles, the digital locations from the reference edgeline of the highway, and the locations of the ground sensor and upstream and downstream ramps serve as input to the software. This software estimates the time that each vehicle passes the ground sensor location X^{sens} and totals the number of vehicles by class passing X^{sens} during a specified time interval. The software code is described fully in Appendix A. We explain the concepts used here and note that comparisons with the manual calculations conducted as described in Merry *et al.* (1996) show that our software works very well.

To control for horizontal curvature of the highway, we use the digitized edgeline of the highway as a linear reference. The program mathematically projects the digitized vehicle locations to this digitized edgeline, providing linear distances from a reference datum. A vehicle that appears in more than one image is automatically identified by its two-part identifier. The linear distance traveled between subsequent imaging of the same vehicle is calculated from the vehicle's locations along the edgeline. This distance is divided by the times between the images to yield an estimate of the vehicle's average velocity U^v in the time between images. The closest imaged location X^v of the vehicle to the ground sensor, the time the vehicle was imaged at this location, the estimated average velocity $U^v(X^v)$ traveled in the time between this image and the next photograph, and the location of the ground sensor X^{sens} are used to estimate the time the vehicle passes the ground sensor. Some vehicles may appear in only one image. These vehicles are assigned velocities equal to the average velocity of the other vehicles in its class – *i.e.*, a car is assigned a velocity equal to the average velocity of all the cars considered on the segment, and a truck is assigned a velocity equal to the average velocity of all the trucks considered on the segment. Once the time that each vehicle passes the ground sensor is estimated, it is straightforward to determine the number of vehicles that pass the sensor during any time interval. Since the identifier indicates the vehicle class, the number of vehicles in each class in any time interval can be readily determined. In this case, the times would correspond to the airplane clock, *i.e.*, the clock that assigns times to the photographs.

Although the process is conceptually straightforward, there are certain controls that must be exerted. The ground sensor data are tagged to ground sensor clocks, while the image-based estimates are tagged to the airplane clock. Discrepancies in these clocks can lead to poor comparisons in a dynamic system such as this. We compensated for time discrepancies by adding or subtracting a constant time offset to the clocks. The details are presented in Merry *et al.* (1996), but the basic approach is to use video data obtained at the site to independently reference the video camera clock to the airplane clock and to the ground sensor clock. An offset is found between the video camera and ground sensor clock that maximizes a correlation measure between video-based estimates of classified counts passing X^{sens} during short intervals and ground sensor-based estimates of classified counts passing X^{sens} during intervals of possibly different durations over a relatively long time period. (We maximized Pearson's correlation factor, obtained video-based estimates of vehicles passing X^{sens} in 5-second intervals, used 1-min intervals for volume-by-length sensors and 1-sec intervals for weigh-in-motion sensors, and compared the estimates over 12-minute periods.) An offset between the video and airplane clocks is found by averaging differences between the video times and estimated photo times when distinguishable vehicles pass X^{sens}. The time offset between the photo and ground sensor clocks is then determined by taking the differences of these photo-video and ground sensor-video time offsets. We expect that we will be able to control for the effect of clock differences more efficiently in tests with real satellite data by simply referencing the ground sensor clocks to the UTC (universal time code) time used in the satellite clocks.

We also control for vehicles entering or exiting the highway. For example, if time intervals analyzed are too long, some vehicles could enter the highway from ramps upstream of the ground sensor after the highway was imaged and pass the ground sensor during the analyzed interval. Similar problems could occur with upstream exit ramps and downstream entrance and exit ramps. Therefore, we limit the time intervals to those such that only vehicles that are imaged downstream of ramps upstream of the ground sensor and upstream of downstream ramps could pass the ground sensor during the time period of analysis. Doing so shortens the lengths of the analyzed intervals from what could otherwise be considered from the imagery, and in some cases we only analyze intervals of less than a minute.

2.3 Results

After compensating for the time discrepancies among the various clocks, we compared volumes-by-class projected as passing the ground sensors from the images, counted from the video, and recorded directly by the ground sensors for estimated concurrent time intervals. We considered the longest time intervals such that vehicles using entrance and exit ramps would not confound the comparisons. That is, we determined the time intervals by estimating the earliest and latest times that imaged vehicles downstream of upstream ramps and upstream of downstream entrance ramps would pass the ground sensors, where upstream and downstream directions are defined with respect to the ground sensor. We shortened the intervals to the nearest minute for the volume-by-length sensors and to the nearest second for the weigh-in-motion based sensors.

The estimated volumes are presented in Table 2.1. In general, the estimates compare favorably with the ground sensor data at the I-70 and I-71 sites and less favorably at the I-270 site, although the I-270 data compare fairly well with the video data. We investigated the I-270 data in more detail and, upon contacting ODOT discovered that the ground sensor (weigh-in-motion) was malfunctioning during the relevant time interval at this site.

Despite the controls for clock differences and the effect of entrance and exit ramps, there could still exist discrepancies between the classification volumes recorded by the ground sensors and those estimated from the images that are not attributable to a failure to detect vehicle classes in the imagery. The discrepancies could result from errors in the estimated vehicle locations, which would cause errors in the X^v and the U^v discussed above. They could also result from the fundamental difference in comparing data taken from images covering a stretch of highway at a point in time to data collected from ground sensors at a point in space during a time interval. In short, if a vehicle would accelerate or decelerate from the estimated speed U^v used to estimate when it would pass X^{sens}, the estimated time of passing X^{sens} would be wrong. Depending on where it fell in the interval of analysis, this could cause discrepancies between the image-estimated volumes and the ground sensor-recorded volumes used as ground truth, even if the vehicles were correctly detected in the images. We have, therefore, begun developing methods and accompanying software to determine upper and lower bounds on the classified volumes that would pass X^{sens} during specified intervals. The bounds would account for reasonable errors in estimated vehicle locations and vehicle acceleration and deceleration characteristics.

Table 2.1. Volumes passing ground sensors estimated from air photos, video and recorded by ground sensors during estimated concurrent time intervals.

	Car Volume			*Truck Volume*		
Segment	*Ground sensor*	*Photo*	*Video*	*Ground sensor*	*Photo*	*Video*
I-70 WB,1996 Time Interval = 6 min	63	72	64	47	42	45
I-70 EB,1996 Time Interval = 2 min	16	16	16	11	12	12
I-270 NB, 1996 Time Interval = 0.83 min	18	31	28	13	6	6
I-270 SB, 1996 Time Interval = 1.58 min	52	45	47	3	10	10
I-71 NB, 1996 Time Interval = 2 min	25	27	26	7	7	5
I-71 SB, 1996 Time Interval = 6 min	52	53	na	32	30	na

We have also begun investigating the contributions of various sources of error in these estimations. Errors due to pixel resolution, digitization of vehicle locations, and projected locations along digitized highway edgelines seem minor. It appears that errors due to estimating time offsets and to the registration of images could be more important. However, in tests using real satellite data, the time offset errors could be reduced by ensuring that the ground sensor clock is calibrated against a UTC clock, which would be the time of the satellite image. The error due to registration of overlapping images should also be reduced because of the precise locations associated with the satellite images. The most important and, perhaps, most irreducible source of error in estimating when vehicles imaged at a given time would pass a ground sensor, appears to be the error in determining the velocity profile of the vehicle between the time that the vehicle is imaged and when it passes the sensor. The bounds we are developing and the accompanying software should help in making useful comparisons between data collected from ground sensors and image-based estimates collected with real satellite data.

Finally, discrepancies between image-estimated and ground sensor-recorded volumes could come from errors in the ground sensors themselves or the classification software used. We mentioned above that we only discovered that the I-270 sensor was malfunctioning upon detailed analysis. We only thought to look at the sensor because of the independent (video) source of data used to form estimates. It actually appears that the image-estimated volumes generally agree with estimates derived from the video data better than with the volumes recorded from the ground sensors. In future feasibility tests, one must, therefore, be careful in considering data collected from ground sensors as ground truth. Obtaining concurrent video data might be necessary when conducting feasibility tests with real satellite data.

Section 3. Image Processing

3.1 Identifying Stationary and Dynamic Pixels

We assume that the remotely sensed image has been segmented for the appropriate highway section. In addition, we assume that we have a historical estimate of the gray-level image of the same highway segment in which all pixels represent the background pavement (stationary pixels). Given a current image of the same highway segment with vehicles, registered appropriately with respect to the background image, we want to detect the pixels corresponding to vehicles. That is, we want to classify pixels in the new image as either stationary (pavement pixels) or dynamic (vehicle pixels). In this section, we present a brief introduction of the statistical pattern recognition procedure we developed to address this task. The technical details are presented in Appendix B. Future development will investigate: (1) how to obtain this initial estimate of the background scene, and (2) classification of clusters of moving pixels, *e.g.*, into cars and trucks (or neither).

Let B_{ij} denote the gray-level of the pixel in row i and column j in the estimated background image of the segment and let Y_{ij} be the gray-level of the same pixel of the current image. A priori, before seeing the new image Y, we start with a prior probability, π_{ij}, on the pixel (i, j) being stationary in the new image. Let

$$\pi_{ij} = \text{Probability that pixel (i,j) is stationary in image Y.} \tag{3.1}$$

In general, the new (current) image, Y, may not have the same overall *brightness* level as the e ted background image, B, due to different lighting conditions under which the two images were acquired. We, therefore, *transform* the brightness level of the background image to make it comparable to match the overall brightness level of the new image Y using a variety of point operations (see, *e.g.*, Castleman (1996), Section 6.3). Let $\phi(B_{ij})$, where ϕ:[0,255]->[0,255] is a brightness adjustment transformation in a specified class of point operations, denote the transformed background image. The parameters of the (unknown) transformation are estimated adaptively from image to image.

Then we obtain the differences, R_{ij}, in gray-level of the current image and the transformed background, *i.e.*,

$$R_{ij} = Y_{ij} - \phi(B_{ij}). \tag{3.2}$$

The stationary pixels in the current image, Y, are expected to have small values of R_{ij}, whereas the dynamic pixels are expected to have R_{ij} that are, in general, large in absolute values. We then estimate the distribution of R, given that the pixels are stationary, p_B, and its distribution, given that the pixels are dynamic, p_V, and compute the *posterior* probability of each pixel being stationary. The posterior probability of a pixel being static is used to classify the pixel into static or dynamic.

The estimation of the transformation ϕ, differencing of the current image and transformed background image, computation of posterior probabilities, and classification of pixels are applied in an iterative manner, until the posterior probabilities converge.

These posterior probabilities can either be used to classify each pixel individually or as input to a rule based clumping procedure. A more advanced statistical pattern recognition procedure, such as a flexible template-matching procedure, which uses the spatial relationship of dynamic pixel clusters could also be used to classify groups of dynamic pixels.

3.2 Overview of the Iterative Procedure

For each pixel in the current image, define the unobservable variables $X_{ij} = 1$ if pixel (i,j) in the image Y is a stationary pixel, and 0 otherwise. Let $\pi_{ij} = \text{Prob}(X_{ij} = 1)$ denote the prior probability that the pixel (i,j) is a background (stationary) pixel.

The conditional distributions of the differences R_{ij} of the background pixels and the vehicle pixels in the current image are defined as follows:

$p(R_{ij} \mid X_{ij} = 1) = p_B(R_{ij})$, probability density of the background pixel differences,
$p(R_{ij} \mid X_{ij} = 0) = p_V(R_{ij})$, probability density of the vehicle/background pixel differences.

Note that $p_B(.)$ should be a unimodal distribution centered at 0, but $p_V(.)$ depends on the gray levels of dynamic pixels in the image Y.

Now the joint density of R and X is given by

$$p(R_{ij}, X_{ij}) = \pi(X_{ij})\, p_B(R_{ij})^{X_{ij}}\; p_V(R_{ij})^{1-X_{ij}}. \tag{3.3}$$

Using Bayes theorem, the posterior probability of $X_{ij} = 1$ is given by

$$p(X_{ij} = 1 \mid R_{ij}) = \frac{p_B(R_{ij})\,\pi_{ij}}{p_B(R_{ij})\pi_{ij} + p_v(R_{ij})(1-\pi_{ij})} \tag{3.4}$$

To be able to compute these posterior probabilities, $p_V(.)$, $p_B(.)$ and $\phi(.)$ all need to be known. In general, these three components in the model are unknown and need to be estimated. A full Bayesian approach would include specifying priors on the unknown components. However, since the amount of information about $p_V(.)$, $p_B(.)$ and $\phi(.)$ is overwhelming (tens of thousands of pixels - a small segment of the size 10 m x 10,000 m has 100,000 1-m pixels), any prior information would likely be swamped by the data. Therefore, the approach adopted here is to estimate $p_V(.)$ and $p_B(.)$ and $\phi(.)$ in an iterative fashion, ignoring the fact that they were estimated when computing the posterior probabilities $p(X_{ij} = 1 \mid R_{ij})$ in each cycle of the iteration. The detailed descriptions of each component of this procedure are given in Appendix B. We illustrate the performance of this procedure for a test image, as well as scanned 1 m x 1 m resolution aerial images in the next section.

3.3 Numerical Study

To illustrate the potential of the methodology described in the previous section, we conducted the following studies. The first study is based on simulated images, while the second uses images formed by scanning air photos taken during our field tests. In the future, we expect to form the background image from an average of several images of the same location. Under light traffic conditions, forming the average would smooth out any signals from vehicles, and the resulting image should be a good approximation of the pavement background. In the studies reported below, we did not have several images of the same location from which to form an average. We, therefore, simulated the background as explained in the studies. The results of both studies show the promise of our method in detecting dynamic pixels that would be associated with vehicles and the robustness of the results to the assumed prior probabilities required by our algorithm.

3.3.1 Simulated Images

To illustrate our approach under a controlled setting, we simulated two images. Specifically, we formed two 30 x 20 images and assumed that all pixels in the images were either static, representing the background pavement, or dynamic, representing vehicles. We assumed that there were two rectangular vehicles of dimensions 5 x 7 and 6 x 8 in the current image, *i.e.*, the image that would be analyzed for vehicle counts. In this way, there were truly 14% (= (5 x 7 + 6 x 8) / (30 x 29) x 100%) dynamic pixels and 86% (= 100% - 14%) background pixels in the current image. The remaining pixels in this current image were assumed to be pavement pixels. The second image was simulated to represent the *background image.* All pixels in this image were assumed to be pavement. We generated gray tones from normal distributions. For the background image gray tones for pixels in columns 4-7, columns 12-16 and columns 19-20, respectively, were generated from N(110,20), N(120,20) and N (80,20) distributions. Gray tones for all other pixels were generated from a N(150,20) distribution. We considered gray tones of pixels in the current image to be the sum of the gray tones in the background image and a N(0,7) disturbance term. We considered the gray tones of the dynamic pixels to be produced by either reflectance off a vehicle or off the pavement covered with a vehicle shadow. The dynamic pixels produced from vehicle reflectance for one vehicle (a darker vehicle) were generated from a N(40,5) distribution. The dynamic vehicle reflectance from the other vehicle (a lighter vehicle) was generated from a N(170,5) distribution. The dynamic shadow reflectance gray tones were generated from a N(0,5) distribution for both vehicles. We regenerated values whenever a negative value or a value greater than 255 was obtained and quantized generated values to the nearest whole number. One realization of the images is shown in Figure 3.1.

We compared our procedure on these images, using 1- and 5-parameter transformations. We also compared these procedures against a variant of a thresholding procedure we had used previously (Merry *et al.,* 1996). When using the transformations, after the procedure has converged, we classified pixels with posterior probabilities greater than 0.5 as dynamic. For the thresholding procedure, we subtracted the gray tones of the pixels in the incoming image from those of the corresponding pixels in the background image. The assumption is that difference values of pixels that were static (pavement) in the two images would be closer to 0 than difference values of pixels that were static (pavement) in one image and dynamic (vehicle) in the other image. Based on this

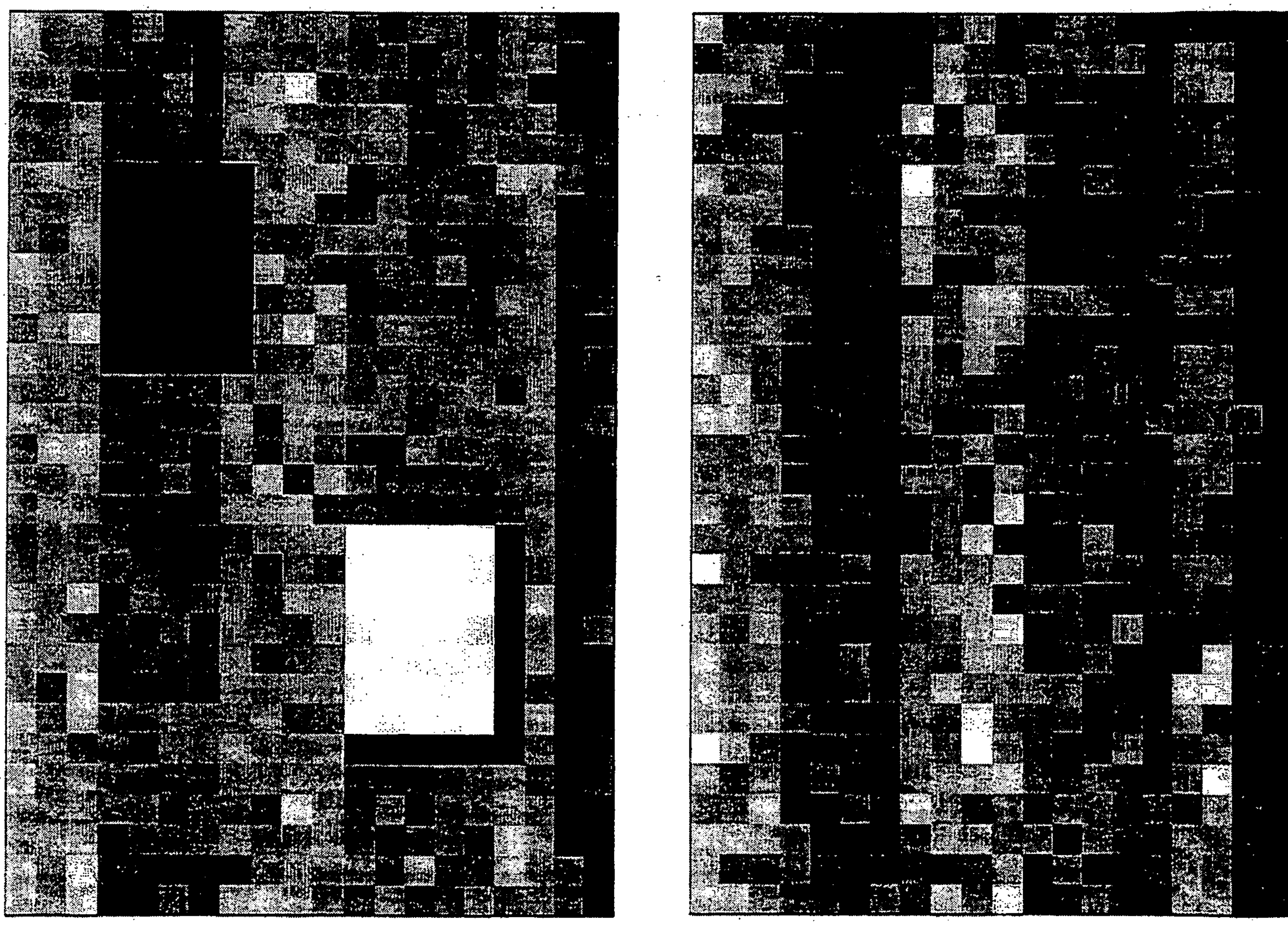


Figure 3.1. Simulated background and the incoming image used in the simulated image study.

assumption, we classified the pixels in the tails as dynamic, where the number of pixels chosen was obtained from the prior estimate of the number of dynamic pixels.

We calculated *errors of omission* and *errors of commission* for each of the procedures. Errors of omission occur when dynamic pixels are not classified as dynamic. Errors of commission occur when pixels that are classified as dynamic are in reality not dynamic. That is, an error of omission occurs when a dynamic pixel is classified as being a background pixel, and an error of commission occurs when a background pixel is classified as being dynamic.

These resulting errors of omission and commission for the three methods assuming three different prior probabilities of dynamic pixels ($1 - \pi_{ij}$, where π_{ij} is defined in eq. 3.1) are presented in Table 3.1. (As mentioned above, 14% of the pixels were truly dynamic in the incoming image. Therefore, this would be the *correct* prior probability that a random pixel would be dynamic.) The results show the superior performance of the transformation methods on this simulated set of images and its robustness across different prior estimates.

Table 3.1 Errors of omission and commission in determining dynamic pixels for three methods on a simulated pair of images, by prior estimate of the percentage of dynamic pixels (true number (%) of dynamic pixels = 83 (14%)).

Prior estimate of dynamic pixels ($1 - \pi_{ij}$) (%)	*Method*	*Errors of omission*	*Errors of commission*
5	Thresholding	52/83 (63%)	0/31 (0%)
	1-parameter transform	15/83 (18%)	0/68 (0%)
	5-parameter transform	3/83 (4%)	1/81 (1%)
15	Thresholding	7/83 (8%)	15/91 (16%)
	1-parameter transform	10/83 (12%)	3/76 (4%)
	5-parameter transform	2/83 (2%)	3/84 (4%)
25	Thresholding	0/83 (0%)	68/151 (45%)
	1-parameter transform	10/83 (12%)	3/76 (4%)
	5-parameter transform	2/83 (2%)	5/86 (6%)

The usual tradeoff between errors of omission and commission is apparent in Table 3.1 for all methods, but it is much less pronounced in the transformation method than in the thresholding method. This tradeoff occurs because the chance of misclassifying a background pixel as dynamic can be reduced by classifying fewer pixels as dynamic, but doing so will increase the chance of omitting a dynamic pixel from being correctly classified as dynamic. If the prior estimates are small or large enough, the thresholding procedure will have very few errors of commission or omission, respectively. In the limit, when the prior estimate goes to 0, no pixels will be classified as dynamic in the thresholding procedure, and there will be no possibility for errors of commission.

This occurs, however, at the expense of a large number of errors of omission, which will go to 100% as the prior estimate goes to 0. On the other hand, as the prior estimate becomes large enough, so many pixels will be classified as dynamic that no dynamic pixels will be omitted. The percentage of errors of omission will go to 0, but the percentage of errors of commission will become very large, as many background pixels will be wrongly classified as dynamic. These extremes are apparent in Table 3.1 for the thresholding procedure. Because of this type of extreme behavior, the thresholding procedure outperforms the transformation method on errors of commission at low (5%) prior estimates. However, the improved performance is only marginal, and the thresholding procedure performs markedly poorly on errors of omission. Similarly, the better performance of the thresholding procedure on errors of omission is overwhelmed by its poor performance on errors of commission at the high (25%) prior estimate.

When considering the errors of omission and commission together, the transformation methods perform much better than the thresholding procedure. Moreover, Table 3.1 indicates that the performance of the transformation methods is not affected much by the prior estimate of the percentage of dynamic pixels. This insensitivity to the prior estimate is encouraging, since it indicates that good results could be produced from even poor estimates of traffic conditions that were present when the image was obtained.

3.3.2 Scanned Images

We also investigated the performance of our method on a pair of air photos scanned to simulate 1-m resolution. We used two overlapping photos taken from I-70. We present these two images, which we call Image A and Image B, in Figure 3.2.

We conducted two experiments on these images. In one we used Image A of Figure 3.2 as the current image, representing the image containing dynamic and static pixels, and formed the background image, representing an image of static pixels, from Image B. In the other experiment, we reversed the roles, using Image A to form the background image and Image B as the current image. To form the background images, we manually replaced the gray values of what we observed to be pixels corresponding to vehicles and their shadows (*i.e.*, the dynamic pixels) with gray values corresponding to the surrounding pavement.

To conduct the experiments, the images had to be registered to a common coordinate system. In both cases we registered the incoming image to that of the background image. Therefore, the registrations were independent in the two experiments. We shall see the effect of imperfect registration below.

We ran the thresholding method and 1-, 2-, and 5-parameter transformations on the images for prior estimates of dynamic pixels ($1 - \pi_{ij}$, where π_{ij} is defined in eq. 3.1) of 1%, 3%, and 7%. (In reality, approximately 3% of the pixels were dynamic.) For each procedure and prior estimate, we calculated errors of omission and commission as we did in the experiments on simulated images in Section 3.3.1.

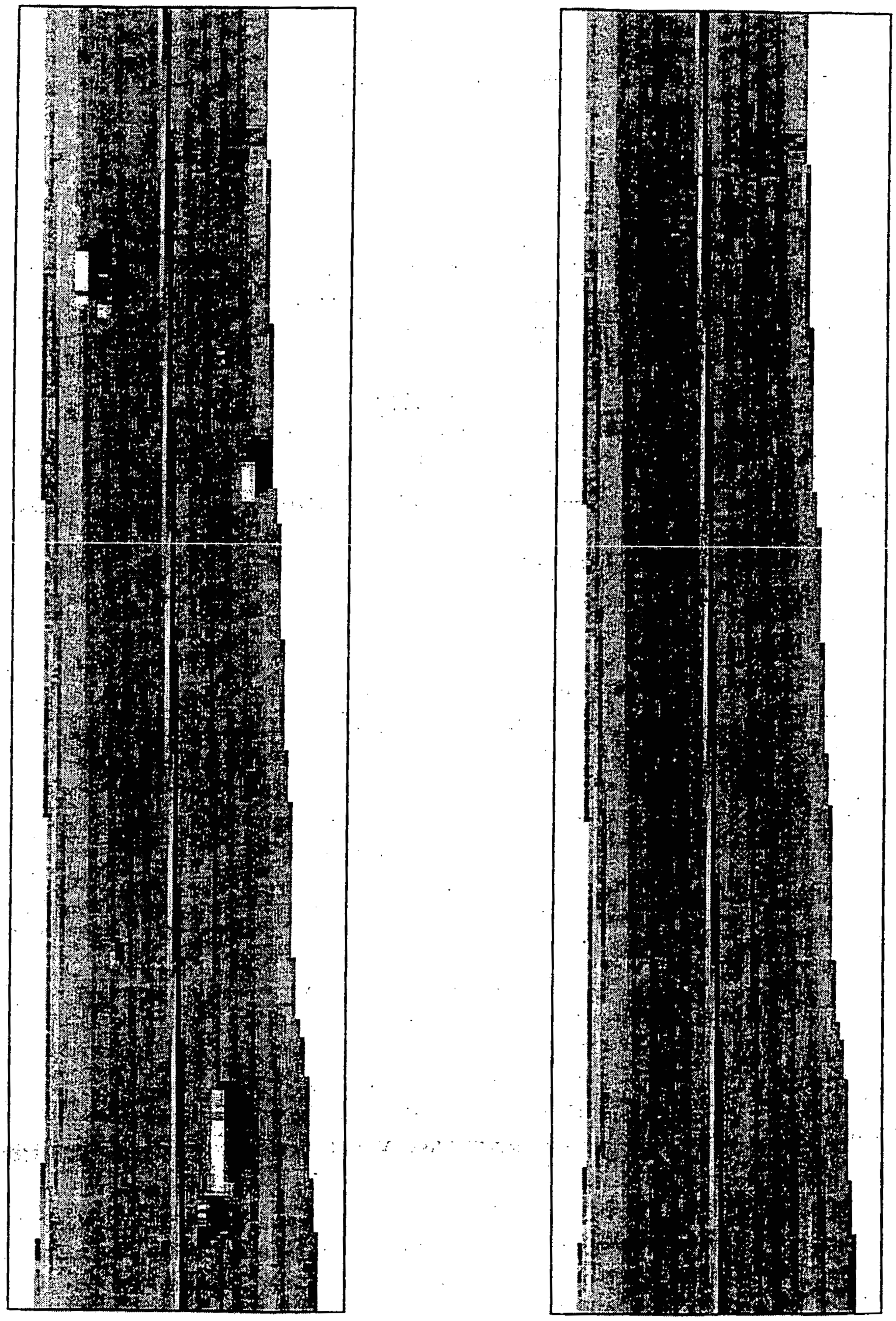


Figure 3.2. Images obtained by scanning air photos to represent 1-m resolution.

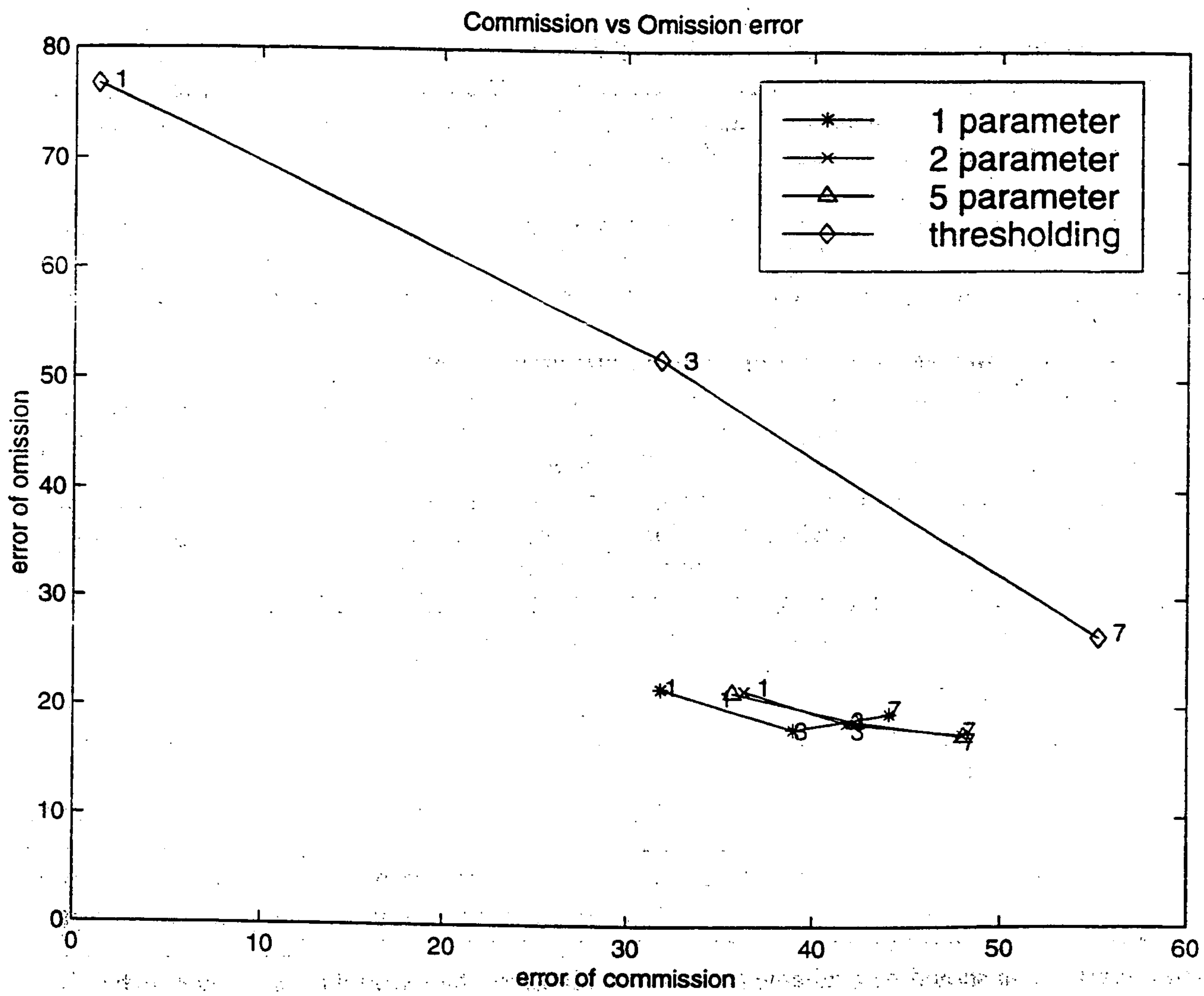
In Figure 3.3, we plot the errors of omission against the errors of commission for the procedures. The numbers next to the plotted points represent the value, in percent, of the prior estimate of dynamic pixels used in the procedures.

Whether image A is used as the current image (Fig. 3.3a) or as the background image (Fig. 3.3b), the transformation procedures are seen to produce fewer errors of omission than the thresholding procedure for any prior estimate of dynamic pixel probability. For 1% and 3% prior estimates of dynamic pixel probability, the thresholding procedure produces markedly fewer errors of commission than the transformation procedures for the corresponding prior dynamic probability estimates. This is not surprising, however, as explained in Section 3.3.1. When prior estimates are so low, the thresholding procedure is expected to produce a low number of commission errors, but it does so at the price of a large number of omission errors.

Moreover, the effect of many of the errors of commission would be reduced when rules, such as those proposed in Merry *et al.* (1996), are applied to determine whether the dynamic pixels are associated with a vehicle or with nonvehicle elements. Images representing the classification of dynamic and static pixels appear in Figures 3.4 and 3.5. In these figures, black pixels are those classified as dynamic, and white pixels are those classified as static. In the transformation images, there are many more isolated pixels being classified as being dynamic. Comparing the images of Figures 3.4 and 3.5 to those of Figure 3.2, one sees that the vehicles correspond to the clumps of dynamic pixels seen in the processed images. The isolated pixels would be eliminated as noise when examined in the context of rules designed to classify groups of dynamic pixels output from the transformation as being vehicle or nonvehicle elements. Moreover, one sees that the shapes of the groups of pixels classified as dynamic in the transformation procedures correspond closely to the shapes of the vehicles seen in Figure 3.2, indicating that vehicle classification rules should perform well when operating on the output of the transformation method.

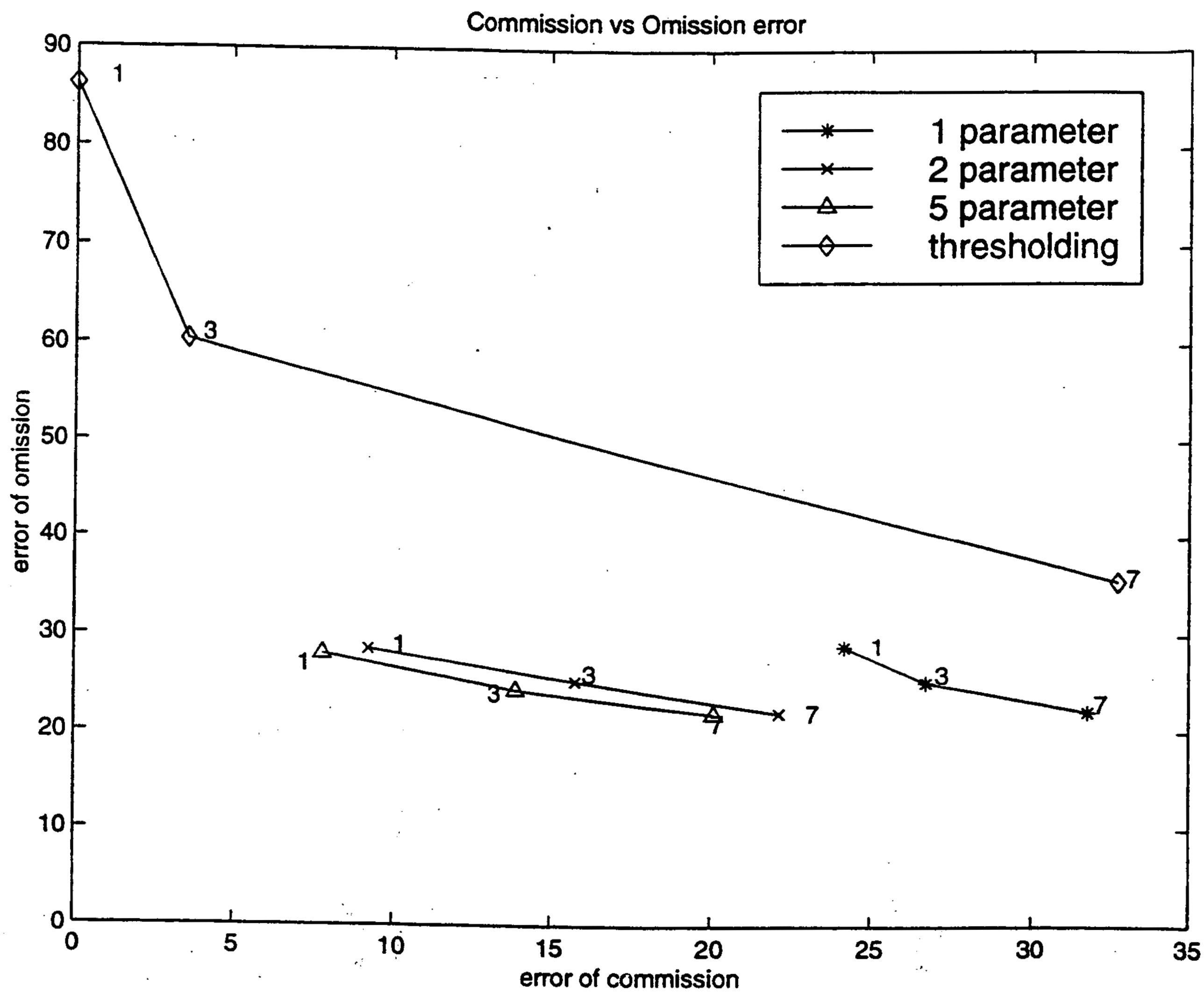
Examination of Figures 3.4 and 3.5 also shows that many errors of commission result from pixels near the median of the highway segment being classified as dynamic. The long, narrow pattern observed would again be conducive to rules correctly classifying the groups of pixels as not being associated with vehicles. Moreover, this phenomenon results in large part from errors in registering the images in the common coordinate system. The median shows up much less in Figure 3.5, where Image B is used as the incoming image, than in Figure 3.4, where Image A is used as the incoming image. (As a result there are many fewer errors of commission in Figure 3.3b than in Figure 3.3a.) We believe that our registration was significantly better in the former case than in the latter case. Better registration should be available from satellite imagery than from the manually registered scanned images used in this study. Still, we expect that the effects of registration will need to be investigated in real satellite images before we feel comfortable in interpreting the outputs of our transformation procedures.

The results again show the robustness of the transformation procedures. Specifically, when the prior estimates vary from 1% to 7%, the thresholding errors of commission and omission vary over ranges of approximately 50% in Figure 3.3a and 35%-40% in Figure 3.3b. The curves produced from the transformation procedures vary over much smaller ranges - approximately 15% and 5%, for errors of commission and omission, respectively, in the two figures. Again, it appears that even rough estimates of traffic conditions when the images are taken can lead to good performance.



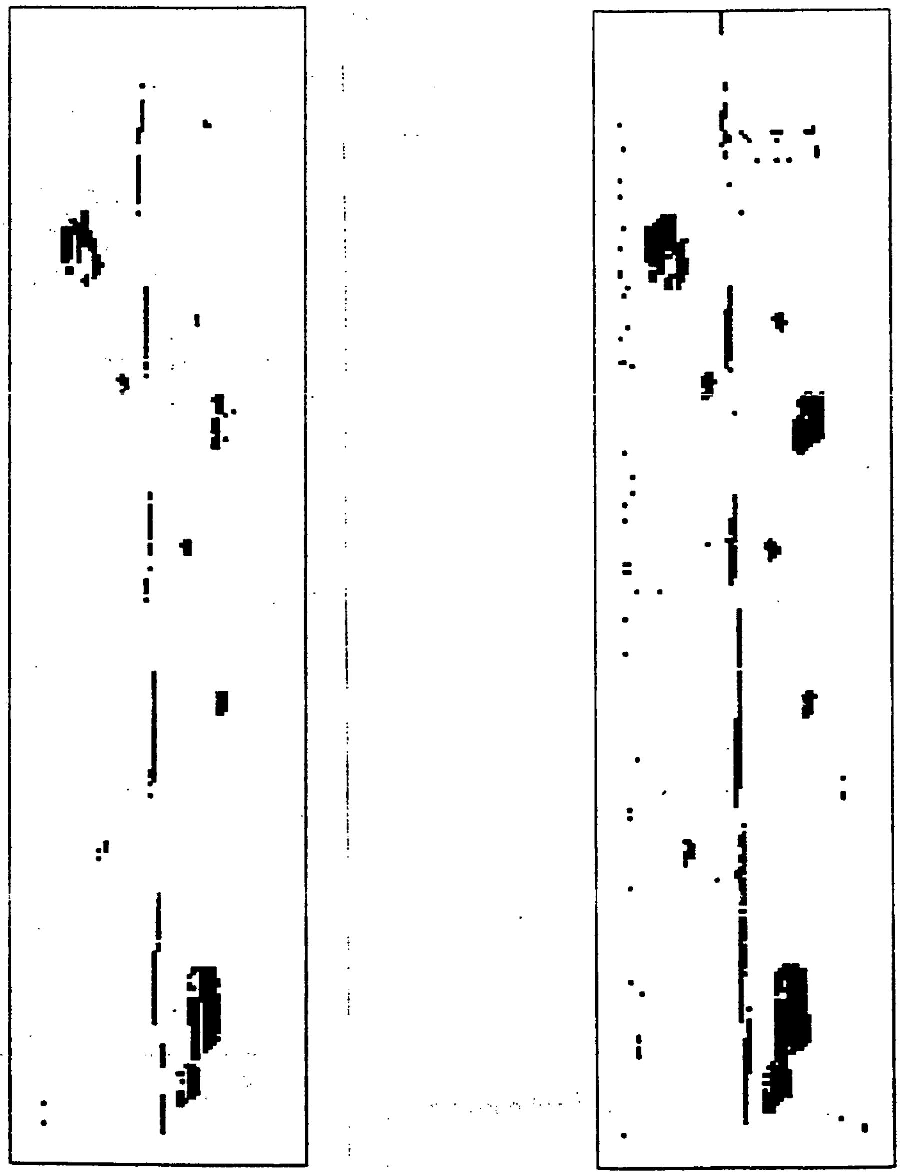
a. Image A.

Figure 3.3. Percent errors of omission *vs.* percent errors of commission in identifying dynamic pixels for the thresholding and transformation (1-, 2-, and 5-parameter) procedures using the images of Figure 3.2, for varying prior estimates of dynamic pixel probabilities (3% dynamic pixels in the image).



b. Image B.

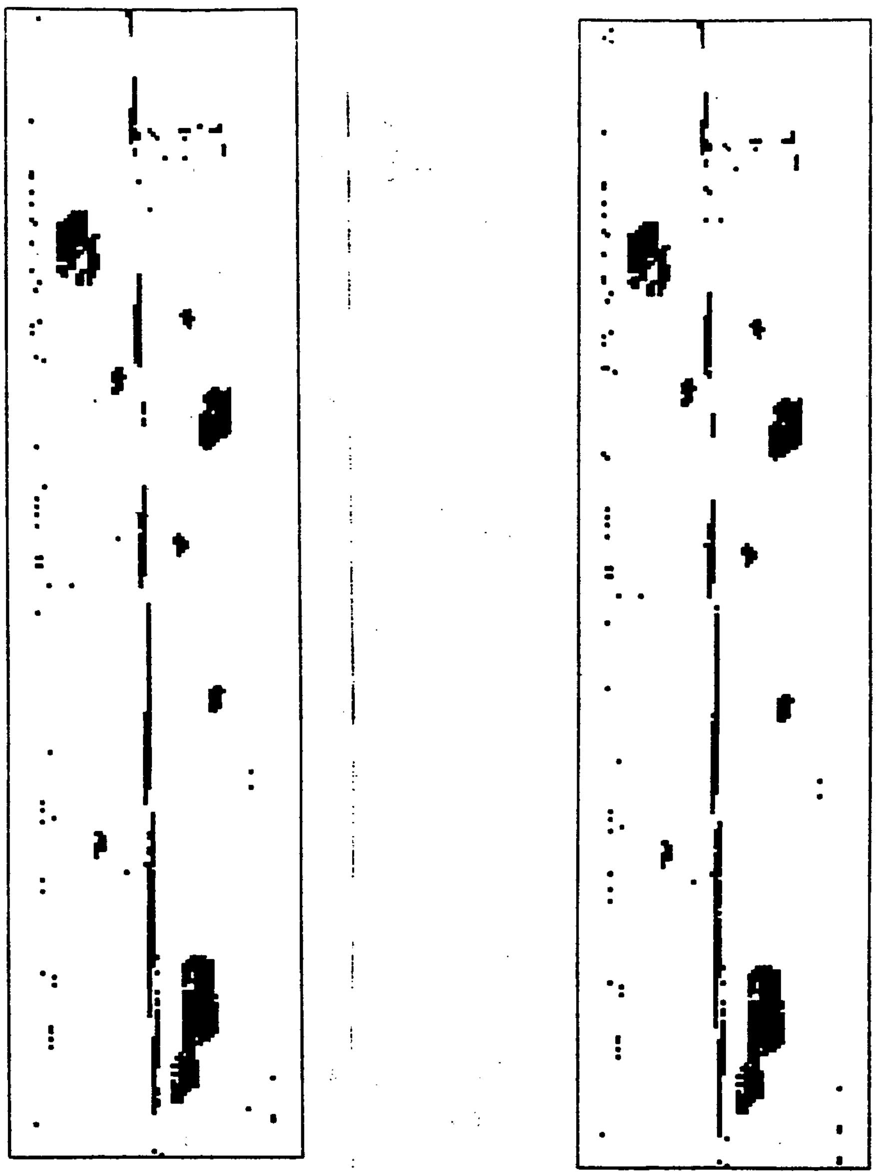
Figure 3.3. Percent errors of omission *vs.* percent errors of commission in identifying dynamic pixels for the thresholding and transformation (1-, 2-, and 5-parameter) procedures using the images of Figure 3.2, for varying prior estimates of dynamic pixel probabilities (3% dynamic pixels in the image).



a. Thresholding.

b. 1-parameter.

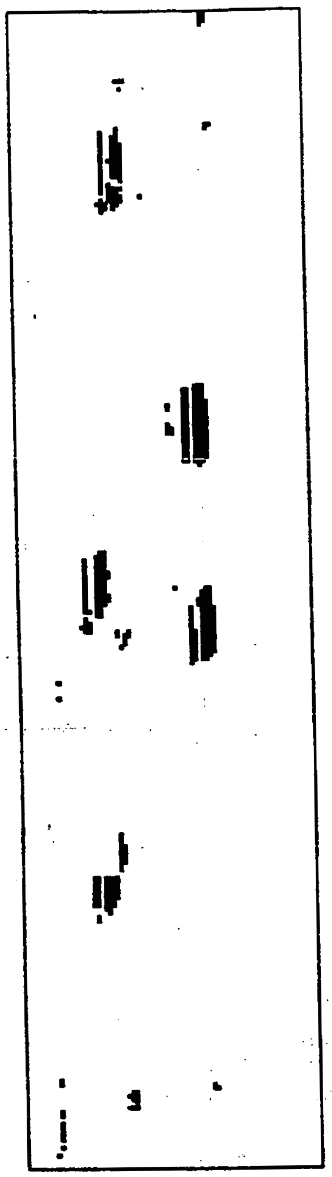
Figure 3.4. Pixels contained in Figure 3.1 classified as dynamic (black) and static (white) for the thresholding procedure and 1-, 2-, and 5-parameter transformations (image A used as incoming image; modified image B used as background image; prior estimate of dynamic pixel probability was 1%).



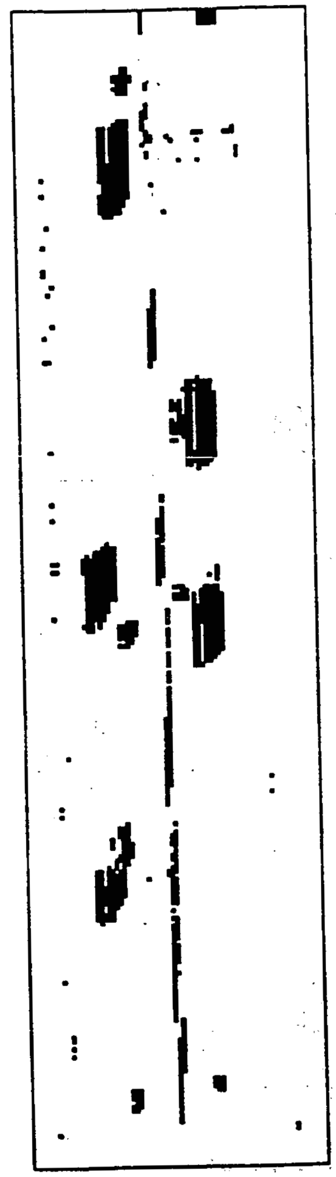
c. 2-parameter.

d. 5-parameter.

Figure 3.4. Pixels contained in Figure 3.1 classified as dynamic (black) and static (white) for the thresholding procedure and 1-, 2-, and 5-parameter transformations (image A used as incoming image; modified image B used as background image; prior estimate of dynamic pixel probability was 1%).

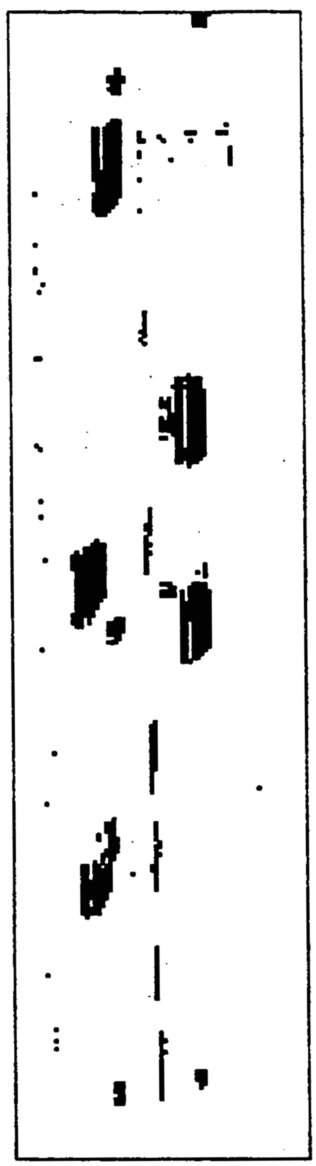


a. Thresholding.

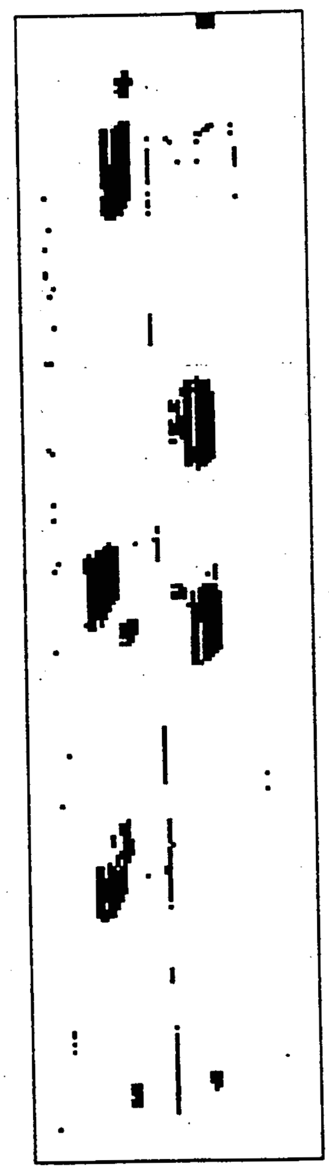


b. 1-parameter.

Figure 3.5. Pixels contained in Figure 3.1 classified as dynamic (black) and static (white) for thresholding procedure and 1-, 2-, and 5-parameter transformations (image B used as incoming image; modified image A used as background image; prior estimate of dynamic pixel probability was 1%).



c. 2-parameter.



d. 5-parameter.

Figure 3.5. Pixels contained in Figure 3.1 classified as dynamic (black) and static (white) for thresholding procedure and 1-, 2-, and 5-parameter transformations (image B used as incoming image; modified image A used as background image; prior estimate of dynamic pixel probability was 1%).

Section 4. Use of Image Data

As mentioned earlier, the satellite data would consist of *snapshots* of the highway segments at instants in time. Several snapshots could be obtained over time, and the greatest benefit in the satellite data might be found in identifying spatial patterns in traffic characteristics. For example, the data might indicate consistently high or low velocities on certain segments. These indications could then be confirmed with traditional spot speed studies. Or, the series of snapshots might show that certain segments exhibit temporal patterns different from those of other segments in the same traffic monitoring sampling class. Aggregate estimates could then be improved by redefining the sampling classes.

Despite these potential advantages, we limit our analysis in this study to the potential of satellite data to improve estimates of Average Annual Daily Traffic (AADT) in homogeneous classes of highway segments. The AADT estimates are used to estimate Vehicle Miles Traveled (VMT) in the class of highway segments, and we also investigate the ability of snapshot data to improve VMT estimates. We base homogeneity of traffic classes on similarity of temporal *expansion factors* described below. We develop computer software to conduct this analysis. Our software contains two main components, a generation component and an estimation component, which we describe in Section 4.1. The generation component simulates true values of AADT and values that would be observed in traffic counting programs. As explained below, we consider 24-hour observations to be representative of data obtained from traditional ground-based sensors and shorter duration observations to be representative of satellite snapshots. The estimation component produces AADT and VMT estimates from the values produced in the generation component.

In Section 4.2, we describe the application of our software to investigate the benefits of combining satellite-based data with ground-based data in the estimation component. The benefits are considered in terms of reduced errors when estimating AADT and VMT, and we investigate the reduction in errors as a function of the number of ground counts, amount of satellite coverage, and variability associated with expanding a satellite snapshot to a daily count.

4.1 Methodology

We consider a highway network consisting of N segments or links with length d_l, $l = 1, 2, \ldots, N$. We specify N as an input to the simulation program, and we randomly generate the link lengths d_l from a truncated normal distribution, $d_l \sim N(\mu_d, \sigma_d)$, $d_l \geq d_{min}$.

4.1.1 Generation of Volume Data

Of the N highway links, we consider that P are equipped with automatic traffic recorders (ATR's) that can count and record daily volumes every day of the year. We also consider that two 24-hour volumes are recorded on M different links with *movable* traffic recorders. The two daily (24-hour) volumes recorded by the movable recorders occur on consecutive days. The parameters P and M determine the supply of ground count information collected and are specified as inputs.

The supply of satellite data is determined by inputs on the time T^R between satellite passes that image links in the network and the number N^I of links imaged each time the satellite passes. Each time the satellite images the area, N^I of the total N links to be imaged are generated at random. A satellite that images with the T^R-day repeat period will produce images of N^I links in the network $365/T^R$ times per year. That is, there will be $365\ N^I/T^R$ link-images produced per year.

An Average Annual Daily Traffic $AADT_l$ is generated for each link $l = 1, 2, \ldots, N$ of the network from a uniform distribution with exogenously input lower and upper bounds, $AADT_{min}$ and $AADT_{max}$. Using the generated true AADT's and randomly generated link lengths, the corresponding value of the true Vehicle Miles Traveled (VMT) is calculated as:

$$\text{VMT} = \sum_{l=1,\ldots,N} d_l * AADT_l. \tag{4.1}$$

AADT's are converted into 24-hour counts for day-of-the-year δ, $\delta \in \{1, 2, \ldots, 365\}$, by calculating a deterministic component U of the 24-volume using day-of-the-week and month-of-the-year expansion factors (McShane and Roess, 1990) and imposing random error on U. Specifically, a set of month-of-the-year or variation expansion factors $\underline{EF}^M = \{EF^M_m,\ m = 1, 2, \ldots, 12\}$ and day-of-the-week expansion factors $\underline{EF}^D = \{EF^D_d, d = 1, 2, \ldots, 7\}$ are specified as input, where, for example, month $m = 1$ corresponds to January, month $m = 2$ corresponds to February, and so on, and day $d = 1$ corresponds to Monday, day $d = 2$ corresponds to Tuesday, and so on,. The factors are chosen so that they would represent expansion of the average volumes on a given month or day to the AADT -- i.e., $(1/12) \sum_{m=1,\ldots,12} (EF^M_m)^{-1} = (1/7) \sum_{d=1,\ldots,7} (EF^D_d)^{-1} = 1.$

The deterministic component of the 24-hour volume for link l on day δ is then:

$$U_{l,\delta} = AADT_l * EF^M_{M(\delta)}{}^{-1} * EF^D_{D(\delta)}{}^{-1}, \tag{4.2}$$

where $AADT_l$ is the AADT of the link l generated as described above, and $M(\delta)$ and $D(\delta)$, respectively, represent the month-of-the-year ($M(\delta) \in \{1,2,\ldots,12\}$) and day-of-the-week ($D(\delta) \in \{1,2,\ldots,7\}$) corresponding to day-of-the year δ ($\delta \in \{1,2,\ldots,365\}$). Multiplying by $EF^M_{M(\delta)}{}^{-1}$ imposes the temporal effect associated with month $M(\delta)$, whereas multiplying by $EF^D_{D(\delta)}{}^{-1}$ imposes the temporal effect associated with day-of-the-week $D(\delta)$.

The 24-hour count on link l on day δ is generated from the det[illegible]stic $U_{l,\delta}$ by considering that the true volume varies from the deterministic model of (4.2) through a specified stochastic model. We use two stochastic models: one uses a log-normally distributed error term; the other generates volumes from a Poisson model (see Appendix C).

4.1.1.1 Log-Normal Generation. We primarily used the "log-normal error term" model in our analysis. In this model, we generate a 24-hour count $V^{(g)}$ that would be observed from a ground sensor (either a permanent ATR or a movable sensor) on link l and day δ as:

$$V^{(g)}_{l,\delta} = U_{l,\delta} * \exp(\varepsilon^{(g)} - \sigma^{(g)2}/2), \tag{4.3}$$

where *exp (.)* is the inverse function of the natural logarithm and $\varepsilon^{(g)} \sim N(0, \sigma^{(g)})$. (This formulation ensures that the expectation of the error term is one, *i.e.*, $E[exp(\varepsilon^{(g)} - \sigma^{(g)2}/2)]=1$.) We assume that $V^{(g)}$ is observed without any measurement error. That is, $V^{(g)}$ is both the true 24-hour volume on link l and day δ and that which is observed from the ground sensor on this link and day.

To simulate the volume estimated from the satellite image, we assume that a satellite image of a link is converted into a 24-hour count $V^{(s)}$ and simulate this 24-hour count as:

$$V^{(s)}_{l,\delta} = U_{l,\delta} * \exp(\varepsilon^{(s)} - \sigma^{(s)2}/2), \tag{4.4}$$

where *exp (.)* is again the inverse function of the natural logarithm and $\varepsilon^{(s)} \sim N(0, \sigma^{(s)})$. (Again, in this formulation the expectation of the error term is one, *i.e.*, $E[exp(\varepsilon^{(s)} - \sigma^{(s)2}/2)]=1$.) The error associated with converting the satellite image into a 24-hour count is handled through the magnitude of $\sigma^{(s)}$ relative to $\sigma^{(g)}$. This process implies that, unlike in the case when generating 24-hour volumes $V^{(g)}$ obtained with ground sensors, the 24-hour counts $V^{(s)}$ estimated from the satellite data are not necessarily the true 24-hour volumes on the segment on the day of observation. We note here that determining the relative magnitudes of $\sigma^{(s)}$ and $\sigma^{(g)}$ to appropriately account for the error in estimating a 24-hour volume from the satellite data is an area for future research. We present our results below as a function of the relative difference in these terms.

4.1.1.2 Poisson Generation. The second stochastic model considers volumes to be Poisson distributed. To generate a 24-hour volume obtained from a ground sensor, we use the deterministic component $U_{l,\delta}$ of Equation (4.2) as the mean of a Poisson distribution for 24-hour volumes and generate the volume from this distribution. That is:

$$V^{(g)}_{l,\delta} \sim \text{Poisson}(U_{l,\delta}). \tag{4.5}$$

Again, the 24-hour volume obtained with the ground-based sensor is assumed to be the true volume in this process.

To generate satellite observations, we simulate a 5-minute volume from a Poisson distribution and convert this generated 5-minute volume to an estimated 24-hour volume. (Time intervals other than five minutes could be used in our program, but we used five minutes as a first approximation of the time interval corresponding to satellite data.) We assume that the 5-minute volume is observed without error, but that there could be error in expanding the 5-minute volume to a 24-hour volume.

To generate the 5-minute volume, we convert the deterministic component of the 24-hour volume $U_{l,\delta}$ of Equation (4.2) to a simulated 5-minute volume obtained in hour h, $h = 1, 2, \ldots, 24$, where, for example, hour $h = 1$ corresponds to 12:00a.m. - 1:00 a.m., $h = 2$ corresponds to 1:00 a.m. 2:00 a.m., and so on. The deterministic component of the 5-minute count U^5_h in hour h is obtained by factoring the 24-hour U by an hourly expansion factor EF^H, taken from a set of exogeneously specified hourly factors $\underline{EF}^H = \{EF^H_h,\ h = 1, 2, \ldots, 24\}$, and converting this hourly volume to a 5-minute count by assuming equal distribution among the twelve 5-minute intervals in the hour:

$$U^5_{l,\delta,h} = U_{l,\delta} * (EF^H_h)^{-1} / 288 \tag{4.6}$$

As with the monthly and daily expansion factors, the hourly expansion factors EF^H_h are specified to represent expansion about average hourly volumes -- i.e., $(1/24) \sum_{h=1,\ldots,24} EF^H_h{}^{-1} = 1$. Dividing by 288 in Equation (4.6) represents the fact that there are 288 5-minute intervals in 24 hours and assumes an equal distribution of a given hour's volume into twelve 5-minute intervals. Unequal distributions could be handled by an expansion factor for subperiods, but since the actual volume will be a randomly generated realization, it would seem overzealous to consider expansion factors for such a short period.

To generate a 5-minute volume $V^{5(s)}_{l,\delta,h}$ obtained in hour h on day δ on link l from a satellite sensor, then, we use the deterministic component $U^5_{l,\delta,h}$ of Equation (4.6) as the mean of a Poisson distribution for 5-minute volumes and generate the volume from this distribution. That is:

$$V^{5(s)}_{l,\delta,h} \sim \text{Poisson}(U^5_{l,\delta,h}). \tag{4.7}$$

We then expand this 5-minute volume to an hourly estimate in hour h by multiplying by 12 and then the hourly estimate to a 24-hour estimate by multiplying by 24 times an "estimate" of the hourly expansion factor $EF^{H\prime}_h$. That is:

$$V^{(s)}_{l,\delta} = 12 * 24 * EF^{H\prime}_h * V^{5(s)}_{l,\delta,h} = 288 * EF^{H\prime}_h * V^{5(s)}_{l,\delta,h}. \tag{4.8}$$

In the work reported here we set $EF^{H\prime}_h$ either equal to the true expansion factor used in generation or to EF^H_h, but future work could investigate the sensitivity of the solution to erroneous estimates of the hourly expansion factor. In this way, the $EF^{H\prime}_h$ value used is not truly an estimate that depends on observations, but an exogenously specified parameter.

4.1.1.3 Output of Data Generation. The simulation program considers one year as the analysis period and uses either Equation (4.3) or Equation (4.5) to generate:

a 24-hour volume count for each of the 365 days of the year for each link assumed to be equipped with a permanent ATR;

two consecutive 24-hour volume counts for each of the links assumed to be covered by a movable ground sensor; these links are randomly generated (without replacement) from the set of links not equipped with permanent ATR's, and it is assumed that the first of the two days that a movable ground sensor collects data on a link is the day after the second of the two days that the sensor collected data on the previously sampled link.

The simulation program also uses either Equation (4.4) or Equation (4.8) to generate:

an estimate of the 24-hour volume for each of N^{I} links randomly generated with replacement every T^{R} days.

One can, therefore, think of partitioning the N links in the simulated network into the following sets based on the types of traffic volumes assumed to be collected on links in the set:

a set **P** consisting of the links that are equipped with permanent ATR's;

a set **MS** consisting of the links for which 24-volumes are obtained from a movable ground sensor during the year *and* for which at least one 24-hour volume estimate is obtained from satellite data during the year;

a set **M** consisting of the links for which 24-hour volumes are obtained from a movable ground sensor but for which no satellite-based 24-hour volume estimates are obtained during the year;

a set **S** consisting of the links for which no ground-based 24-hour volumes are obtained, but for which at least one satellite-based 24-hour volume estimate is obtained during the year;

a set **R** consisting of the links for which neither ground-base nor satellite-based 24-hour volumes are obtained during the year.

We call N_P, N_{MS}, N_M, N_S, and N_R, the numbers of links in the respective sets, with $N_P + N_{MS} + N_M + N_S + N_R = N$. We also assume that the links have been renumbered so that the first N_P links are those in set **P**, the next N_{MS} links are those in set **MS**, the next N_M links are those in set **M**, the next N_S links are those in set **S**, and the final N_R links are those in set **R**. In this way, the output of the simulation program consists of "ground based" and "satellite-based" data. The ground-based data are comprised of:

$$V^{(g)}_{l,\delta}, \qquad \delta = 1, 2, \ldots, 365; \; l = 1, 2, \ldots, N_P;$$
$$V^{(g)}_{l,\delta} \qquad \delta = \Delta g(l), \Delta g(l)+1; \quad l = N_P+1, N_P+2, \ldots, N_P+N_{MS}+N_M;$$

where $\Delta g(l)$ indicates the day on which the first of the two consecutive 24-hour ground-based counts are obtained with movable ground sensors.

The satellite-based data are comprised of:

$$V^{(s)}_{l,\delta}, \qquad \delta = \Delta s_1(l), \Delta s_2(l), \ldots, \Delta s_{Il}(l); \quad l = N_P+1, N_P+2, \ldots, N_P+N_{MS}, N_P+N_{MS}+N_M+1, \ldots, N_P+N_{MS}+N_M+N_S$$

where $\Delta s_i(l)$ indicates the day on which a satellite-based estimated daily volume was produced on link l for the i^{th} time in the year, and Il indicates the number of times that a satellite-based estimated volume was produced on link l during the year.

The simulation program also produces the true values of the AADT's and link lengths for each link l and the true VMT as output, *i.e.*:

d_l, $\qquad l = 1, \ldots, N,$

AADT_l, $\qquad l = 1, \ldots, N,$

VMT.

A listing of the generation programs can be found in Appendix D.

4.1.2 Estimation of Traffic Parameters

Our estimation programs use the output of the simulation programs as input and estimate Annual Average Daily Traffic (AADT) for each link l in the network and then Vehicle Miles Traveled (VMT) from these AADT's and the corresponding segment lengths d_l. We consider two methods – what we call the *traditional method* and what we call a *model-based method* – to produce these estimates. We produce estimates when using only the ground-based data and when combining the ground-based and satellite-based data.

4.1.2.1 Traditional Esti n Method

Ground-based data only: Estimating AADT's using the traditional method with only ground-based data is similar to the commonly proposed method (U.S. Department of Transportation 1992, McShane and Roess 1990) of:

i) estimating expansion factors from data obtained from permanent ATR's;
ii) using these expansion factors to convert 24-hour volumes into annual average estimates;
iii) averaging the different annual average estimates for the same link to produce an estimate of that link's AADT;
iv) estimating AADT on links with no observations from the AADT estimates of the links for which there were observations.

Specifically, the AADT's for the N_P links in set **P** equipped with permanent ATR's are estimated as the average of the 365 hourly volumes:

$$AADT^{(g)}_l = \sum_{\delta=1,\dots,365} V^{(g)}_{l,\delta} / 365, \qquad l = 1, \dots, N_P; \tag{4.9}$$

where we append "$^{(g)}$" to the AADT variable to indicate that the AADT is estimated from ground data only.

The 365 $V^{(g)}$ volumes on these N_P links are also used to estimate the month-of-year and day-of-week expansion factors as:

$$EF^{(g)M}_m = [AADT^{(g)}_l / < V^{(g)}_{l,\delta} >_{M(\delta)=m}]_{l \in \{1, \dots, NP\}}, \qquad m = 1, 2, \dots, 12; \tag{4.10}$$
$$EF^{(g)D}_d = [AADT^{(g)}_l / < V^{(g)}_{l,\delta} >_{D(\delta)=d}]_{l \in \{1, \dots, NP\}}, \qquad d = 1, 2, \dots, 7; \tag{4.11}$$

where $[\,.\,]_{l \in \{1, \dots, NP\}}$ represents the harmonic average over the N_P segments with permanent ATR's, and $<.>_{M(\delta)=m}$ and $<.>_{D(\delta)=d}$ represent the arithmetic averages over all days-of-the-year δ that are, respectively, in month m and on day-of-the-week d, and the "$^{(g)}$" 's appended to the EF's indicate that the factors are estimated from ground-based data only.

The AADT's using ground-based data only for the links in sets **MS** and **M** where counts have been taken with movable ground sensors are estimated as:

$$AADT^{(g)}_l = \sum_{\delta=\Delta g(l),\, \Delta g(l)+1} V^{(g)}_{l,\delta} * EF^{M}_{M(\delta)} * EF^{D}_{D(\delta)} / 2, \qquad l = N_P+1, \dots, N_P+N_{MS}+N_M, \tag{4.12}$$

that is, the average of the two 24-hour volumes obtained on the link on consecutive days ($\Delta g(l)$ and $\Delta g(l)+1$) after "expanding" the 24-hour volume into an estimate of the annual average using the appropriate monthly and day-of-the-week expansion factors.

The AADT's estimated when using only ground-based data for the links in sets **S** and **R**, where no ground-based data have been obtained, are estimated as the arithmetic average of the estimated AADT's of the links for which ground-based data have been obtained:

$$AADT^{(g)}_l = \sum_{k=1,\dots,NP+NMS+NM} AADT^{(g)}_k / (N_P+N_{MS}+N_M), \qquad l = N_P+N_{MS}+N_M+1, \dots, N. \tag{4.13}$$

The VMT using ground-based data only $VMT^{(g)}$ is estimated as:

$$VMT^{(g)} = \sum_{l=1,\dots,N} d_l * AADT^{(g)}_l. \tag{4.14}$$

Combined satellite-based and ground-based data: When combining the satellite-based data with the ground-based data in the traditional method, we treat 24-hour volumes generated from simulated satellite observations in the same way that we treat 24-hour volumes generated from simulated ground observations, except when simulated satellite-based estimates occur on one of the N_P links assumed to have permanent ATR's. In this case, we ignore the satellite observation, since the ground-based data on the links equipped with permanent ATR's are assumed to be error-free data.

Specifically, the AADT's for the N_P links of set **P** equipped with permanent ATR's are estimated from ground data only, so that:

$$AADT^{(sg)}_l = AADT^{(g)}_l, \qquad l = 1, \ldots, N_P, \qquad (4.15)$$

where $AADT^{(g)}_l$ is determined from Equation (4.9), and we now use "$^{(sg)}$" to indicate that we are considering the case where we can combine the satellite-based data with the ground-based data to produce estimates. The month-of-year and day-of-week expansion factors are again estimated using the ground-based data on the links assumed to be equipped with permanent ATR's so that;

$$EF^{(sg)M}_m = EF^{(g)M}_m, \qquad m = 1, \ldots, 12; \qquad (4.16)$$
$$EF^{(sg)D}_d = EF^{(g)D}_d, \qquad d = 1, \ldots, 7. \qquad (4.17)$$

where $EF^{(g)M}_m$ and $EF^{(g)D}_d$, respectively are determined from Equations (4.10) and (4.11).

For the N_{MS} links on which 24-hour volumes are observed with a movable ground sensor and for which at least one satellite observation is obtained during the year – *i.e.*, the links in set **MS** – the 24-hour volumes (whether obtained from the ground sensor or estimated from the satellite observation) are expanded to an estimate of the annual average using the appropriate expansion factors and then averaged. That is:

$$AADT^{(sg)}_l = \Big(\sum_{\delta=\Delta g(l),\, \Delta g(l)+1} V^{(g)}_{l,\delta} * EF^M_{M(\delta)} * EF^D_{D(\delta)} + \sum_{\delta=\Delta s1(l),\ldots,\, \Delta sIl\,(l)} V^{(s)}_{l,\delta} * EF^M_{M(\delta)} * EF^D_{D(\delta)} \Big) / (2 + I_l),$$
$$l = N_P+1, \ldots, N_P+N_{MS}, \qquad (4.18)$$

where the average is seen to be taken over the 2 ground-based observations and the I_l satellite-based observations.

The AADT's for the set **M** of links simulated to have ground-based observations taken from a movable ground sensor but for which no satellite data are obtained are estimated from the ground-based data only as in Equation (4.12). When only considering ground-based data, Equation (4.12) was used to estimate AADT's for all $N_{MS}+N_M$ links where ground-based data were obtained with movable sensors. The equation would only be used for the N_M links in set **M** when considering combined satellite-based and ground-based data. That is:

$$AADT^{(sg)}_l = AADT^{(g)}_l, \qquad l = N_P+N_{MS}+1, \ldots, N_P+N_{MS}+N_M, \qquad (4.19)$$

where $AADT^{(g)}_l$ is determined in Equation (4.12).

The AADT's for the set **S** of links for which no ground-based data were simulated, but for which at least one satellite observation is obtained during the year are estimated as the average of the expanded satellite-based estimates of the 24-hour volumes. That is:

$$AADT^{(sg)}_l = \sum_{\delta=\Delta s1(l),\ldots,\Delta sIl(l)} V^{(s)}_{l,\delta} * EF^M_{M(\delta)} * EF^D_{D(\delta)} \,/\, I_l, \quad l = N_P+N_{MS}+N_M+1, \ldots, N_P+N_{MS}+N_M+N_S, \quad (4.20)$$

where the average is seen to be taken over the I_l satellite-based observations.

Finally, as before, the AADT's of links for which no data are available – *i.e.*, the links in set **R** – are estimated as the arithmetic average of the estimated AADT's of the links for which some data have been simulated:

$$AADT^{(sg)}_l = (\sum_{k=1,\ldots,NP+NMS+NM+NS} AADT^{(sg)}_k \,/\, (N_P+N_{MS}+N_M+N_S), \quad l = N_P+N_{MS}+N_M+N_S+1, \ldots, N. \quad (4.21)$$

When combining ground-based and satellite-based data, the VMT, now denoted $VMT^{(sg)}$, is estimated as:

$$VMT^{(sg)} = \sum_{l=1,\ldots,N} d_l * AADT^{(sg)}_l. \quad (4.22)$$

A listing of the traditional method estimation code is provided in Appendix D.

4.1.2.2 Model-Based Estimation Method

Ground-based data only: When assuming the log-normal error model as that which generates the link volumes, our model-based method uses a least squares approach to estimate AADT's. Unlike the traditional estimation method, the model-based model uses all observations to estimate the parameters of the model assumed to produce the observations.

Specifically, when using ground-only data the model-based method assumes that Equations (4.2) and (4.3) produce observed link volumes. Substituting Equation (4.2) into Equation (4.3) and taking the natural logarithm of both sides produces:

$$\ln V^{(g)}_{l,\delta} = \ln AADT_l - \ln EF^M_{M(\delta)} - \ln EF^D_{D(\delta)} - \sigma^{(g)2}/2 + \varepsilon^{(g)}_{l,\delta}, \quad \delta = 1,\ldots,365 \quad l = 1, \ldots, N_P \quad (4.23a)$$

for the N_P links in set **P**, and

$$\ln V^{(g)}_{l,\delta} = \ln AADT_l - \ln EF^M_{M(\delta)} - \ln EF^D_{D(\delta)} - \sigma^{(g)2}/2 + \varepsilon^{(g)}_{l,\delta}, \quad \delta = \Delta g(l), \Delta g(l)+1; \quad l = N_P+1, \ldots, N_P+N_{MS}+N_M; \quad (4.23b)$$

for the $N_{MS}+N_M$ links in sets **MS** and **M**. These $365N_P+2(N_{MS}+N_M)$ equations are used in a least squares routine to minimize the sum of the squares of the $\varepsilon^{(g)}_{l,\delta}$ terms and produce estimates of the $(N_P+N_{MS}+N_M)$ $\ln AADT_l$'s, the 12 $\ln EF^M_{M(\delta)}$'s, the 7 $\ln EF^D_{D(\delta)}$'s, and

$\sigma^{(g)2}/2$. We denote the estimated values of the ln AADT's by $ln\ AADT^{(g)}{}'$. Unbiased AADT estimates $AADT^{(g)}{}_l$ can be shown to be:

$$AADT^{(g)}{}_l = \exp(ln\ AADT^{(g)}{}'_l - \tau_l{}^2/2)\,, \qquad l=1,\ldots, N_P+N_{MS}+N_M; \qquad (4.24)$$

where $\tau_l{}^2$ is the (estimated) variance of the $ln\ AADT^{(g)}{}'_l$ estimate.

Unbiased estimates $AADT^{(g)}{}_l$ of the AADT's on the N_S+N_R links where no ground data were obtained can similarly be shown to be:

$$AADT^{(g)}{}_l = \exp(<ln\ AADT^{(g)}{}'_\xi>_{\xi=1,\ldots,NP+NMS+NM} - \mathrm{Var}<lnAADT^{(g)}{}'_\xi>_{\xi=1,\ldots,NP+NMS+NM}/\,2)\,,$$
$$l=N_P+N_{MS}+N_M+1,\ldots, N; \qquad (4.25)$$

where $<ln\ AADT^{(g)}{}'_\xi>_{\xi=1,\ldots,NP+NMS+NM}$ and $Var<\ln AADT^{(g)}{}'_\xi>_{\xi=1,\ldots,NP+NMS+NM}$, respectively, represent the arithmetic average and variance of the average of the estimated "ln AADT's" for the links in sets **P**, **MS**, and **M** output from the least squares routine.

The estimated VMT using ground data only $VMT^{(g)}$ is then computed as:

$$VMT^{(g)} = \Sigma_{\,l=1,\ldots,N}\ d_l * AADT^{(g)}{}_l. \qquad (4.26)$$

where the $AADT^{(g)}{}_l$ values are determined from Equations (4.24) or (4.25), and the d_l values were generated in the simulation program.

Combined satellite-based and ground-based data: When assuming the log-normal error model and combining satellite and ground data, the model-based method parallels that described when using ground-based data alone and assuming the log-normal error model. Equations (4.2) and (4.3) are again assumed to produce 24-hour link volumes that are observed by ground-based sensors, and Equations (4.2) and (4.4) are assumed to produce 24-hour estimated link volumes derived from satellite observations. Therefore, in addition to Equations (4.23a) and (4.23b), the satellite-based data can be used with Equations (4.2) and (4.4) to produce:

$$\ln V^{(s)}{}_{l,\delta} = \ln AADT_l - \ln EF^M{}_{M(\delta)} - \ln EF^D{}_{D(\delta)} - \sigma^{(s)2}/2 + \varepsilon^{(s)}{}_{l,\delta},$$
$$\delta = \Delta s_1(l),\ldots, \Delta s_{Il}; \quad l = N_P+1,\ldots, N_P+N_{MS};$$
$$N_P+N_{MS}+N_M+1,\ldots,N_P+N_{MS}+N_M+N_S; \qquad (4.27)$$

The $365N_P+2(N_{MS}+N_M)$ equations associated with the ground-based data (Equations (4.23a) and (4.23b)) and the $\Sigma_{l=NP+1,\ldots,NP+NMS}(I_l) + \Sigma_{l=NP+NMS+NM+1,\ldots,NP+NMS+NM+NS}(I_l)$ equations associated with the satellite-based data (Equations (4.27)) are used in a weighted least squares routine (Chambers and Hastie, 1992) to minimize the (weighted) sum of the squares of the $\varepsilon^{(g)}{}_{l,\delta}$ and $\varepsilon^{(s)}{}_{l,\delta}$ terms to produce estimates of the $(N_P+N_{MS}+N_M+N_S)$ $ln\ AADT_l$'s, the 12 $ln\ EF^M{}_{M(\delta)}$'s, the 7 $ln\ EF^D{}_{D(\delta)}$'s, $\sigma^{(g)2}/2$, and $\sigma^{(s)2}/2$. (The weights used in the routine are inversely proportional to the variances $\sigma^{(g)2}$ and the $\sigma^{(s)2}$, which are assumed to be known as inputs for the routine in this preliminary work. In reality, these variances would be unknown – indeed, they are estimated in the routine, as seen in Equations (4.23) and (4.27). A process could be developed that iterates until the variances assumed when dete g the input weights are close to those that are estimated from the routine.)

We now denote the estimated values of the ln AADT's by $ln\ AADT^{(sg)}{}'$ to indicate that both satellite and ground data have been used in this estimate. Similar to what we did above, we form the unbiased AADT estimates as:

$$AADT^{(sg)}{}_l = \exp(ln\ AADT^{(sg)}{}'_l - \tau_l^2/2), \qquad l=1,\ldots, N_P+N_{MS}+N_M+N_S; \qquad (4.28)$$

where τ_l^2 is, again, the (estimated) variance of the estimated $ln\ AADT^{(sg)}{}'_l$.

The unbiased estimates of the AADT's on the N_R links in set **R** where no ground data were obtained are, then:

$$AADT^{(sg)}{}_l = \exp(\langle ln\ AADT^{(sg)}{}'_\xi\rangle_{\xi=1,\ldots,NP+NMS+NM+NS} - \mathrm{Var}\langle lnAADT^{(sg)}{}'_\xi\rangle_{\xi=1,\ldots,NP+NMS+NM+NS}/2), \quad l=N_P+N_{MS}+N_M+N_S+1,\ldots, N; \quad (4.29)$$

where $\langle ln\ AADT^{(sg)}{}'_\xi\rangle_{\xi=1,\ldots, NP+NMS+NM+NS}$ and $Var(\langle ln\ AADT^{(sg)}{}'_\xi\rangle_{\xi=1,\ldots, NP+NMS+NM+NS}$, respectively, represent the arithmetic average and variance of the average of estimated "log AADT's" for the links in sets **P**, **MS**, **M**, and **S** output from the least squares routine.

The estimated VMT using combined satellite and ground data $VMT^{(sg)}$ is then computed as:

$$VMT^{(sg)} = \sum_{l=1,\ldots,N} d_l * AADT^{(sg)}{}_l. \qquad (4.30)$$

where the $AADT^{(sg)}{}_l$ values are determined from Equations (4.28) or (4.29), and the d_l values were generated in the simulation program.

We developed but did not implement the underlying theory of the methodology for model-based estimation when assuming volumes were generated from a Poisson distribution. That is, when assuming Poisson generation, we only used the traditional model.

A listing of the model-based estimation code is presented in Appendix E.

4.2 Numerical Study

We ran our simulation program for several sets of input values. In all cases, we considered a network with $N = 100$ links and $N_P = 3$ links; *i.e.*, we assumed that 3% of the links were equipped with permanent ATR's, a percentage roughly equal to that in the Ohio Department of Transportation system. We generated the link lengths d_l from a truncated normal distribution with $\mu_d = 1.5$, $\sigma_d = 1.0$, and $d_{min} = 0.3$, and the true link AADT's from a uniform distribution with $AADT_{min} = 10{,}000$ vehicles and $AADT_{max} = 90{,}000$ vehicles (see Section 4.1 and Table 4.1). We set the variance of the error term of Equation (4.3) $\sigma^{2(g)} = 0.04$ and the satellite repeat period at $T^R = 18.25$ days.

We considered different numbers of movable ground sensors, variance of the error term associated with satellite data in Equation (4.4), and number of links imaged by the satellite per repeat period. Specifically, we considered combinations of $N_M = 0, 12, 25, 38, 50$; $\sigma^{2(s)} = 0.04, 0.16, 0.36$, and $N^I = 5, 10, 15$. In McCord *et al.* (1995) we estimated that a 1-m resolution satellite would be capable of imaging roughly 0.5% of the links in the continental United States per day. This percentage accounts for the fact that images could not be obtained in cloudy conditions or at nighttime. Therefore, a 1-m sensor on a satellite platform would be capable of imaging $365*0.5\%$ of the $N=100$ network links per year. Since the satellite is assumed to image N^I links each of the $365/T^R$ times per year it repeats its coverage of the region, we can consider the "equivalent satellite coverage" ESC as:

$$\begin{aligned} ESC &= N^I * (365 / T^R) / (365 * 0.005 * N) = 200 * (N^I/N) / T^R \\ &= 200 * (N^I/100) / 18.25 = N^I / 9.125. \end{aligned} \qquad (4.31)$$

This equivalent satellite coverage represents the fraction of data from a 1-m resolution sensor equivalent to that which would be produced with the assumed N^I and T^R values. For example, $N^I = 5$ links would correspond to using roughly half (*i.e.*, $ESC = 5/9.125 \approx 0.5$) of the data produced from a 1-m sensor on a satellite platform.

We summarize these input parameters and the expansion factors used in Table 4.1.

For each set of input values, we ran the simulation-generation program 100 times, simulating 100 independent replications of a one-year analysis period. Each run produced for each link l one true AADT, one AADT estimated when using the ground-based data only, and one AADT estimated from combined ground-based and satellite-based data. As above, we denote these values $AADT_l$, $AADT^{(g)}_l$, and $AADDT^{(sg)}_l$, respectively. We formed the relative AADT error for link l for each simulation run r when either using ground-based data only or when combining satellite-based data with ground-based data as $(AADT^{(\cdot)}_{l,r} - AADT_{l,r}) / AADT_{l,r}$, $^{(\cdot)} = {}^{(g)}, {}^{(sg)}$; $l = 1, \ldots, N$; $r = 1, \ldots, 100$; and the root mean squared relative error in AADT across all links for a given simulation run as r:

$$RMSREaadt^{(\cdot)}_r = (\Sigma_{l=1,\ldots,N} ((AADT^{(\cdot)}_{l,r} - AADT_{l,r}) / AADT_{l,r})^2)^{0.5}, \quad {}^{(\cdot)} = {}^{(g)}, {}^{(sg)}; \; r = 1, \ldots, 100; \qquad (4.32a)$$

From these 100 values, we formed the average of the root mean squared relative errors across all runs as:

$$ARMSREaadt^{(\cdot)} = \Sigma_{r=1,\ldots,100} \, RMSREaadt^{(\cdot)}_r / 100, \qquad {}^{(\cdot)} = {}^{(g)}, {}^{(sg)}. \qquad (4.32b)$$

Table 4.1 Values of input parameters used in simulation-estimation runs.

Parameter name (notation)	*Values used in program*
Number of total network links (N)	100
Number (%) of links equipped with permanent ATR's (N_P)	3 (3%)
Number (%) of links equipped with movable ground sensors (N_M)	0 (0%), 12 (12%), 25 (25%), 38 (38%), 50 (50%)
Mean of link length distribution (μ_d)	1.5
Standard deviation of link length distribution (σ_d)	1.0
Lower bound of link length distribution (d_{min})	0.3
Upper bound of AADT distribution ($AADT_{max}$)	10,000
Lower bound of AADT distribution ($AADT_{min}$)	90,000
Day-of-the-week expansion factors ($EF^D = \{EF^D_d, d = 1,...,7\}$)	{1.072000, 1.121000, 1.108000, 1.098000, 1.015000, 0.899000, 0.790976}
Month-of-the-year expansion factors ($EF^M = \{EF^M_m, m = 1,..., 12\}$)	{1.215000, 1.191000, 1.100000, 0.992000, 0.949000, 0.918000, 0.913000, 0.882000, 0.884000, 0.931000, 1.026000, 1.152032}
Hourly expansion factors ($EF^H = \{EF^H_h, h = 1,..., 24\}$) *(used for Poisson generation)*	{1.011000, 1.123000, 1.221000,1.709000, 2.062000, 1.532000, 0.925000, 0.703000, 0.331000, 0.433000, 0.825000, 0.995000, 1.601000, 1.774000, 0.964000, 0.734000, 0.402000, 0.373000, 0.854000, 1.437000, 1.755000, 2.158000, 2.105000, 1.123191}
Variance of "ground-based data error term" ($\sigma^{2(g)}$)	0.04
Variance of "satellite-based data error term" ($\sigma^{2(s)}$)	0.04, 0.16, 0.36
Satellite repeat period (T^R)	18.25 days
Number of links imaged per pass (N^I)	5, 10, 15
Approximate equivalent satellite coverage (*ESC*) (determined by T^R and N^I)	0.5, 1.0, 1.5

We used the generated link lengths to estimate the true VMT, the VMT estimated when using ground-based data only, and the VMT estimated when combining satellite-based with ground-based data as in Equations (4.1), (4.14), and (4.22). Somewhat similar to what we did in summarizing the AADT errors, we formed the relative VMT error for a given run r and the average relative errors across all runs, respectively, as:

$$REvmt^{(\cdot)}_r = | VMT_r - VMT^{(\cdot)}_r | / VMT_r , \qquad {}^{(\cdot)} = {}^{(g)}, {}^{(sg)}; \ r = 1, ...,100; \qquad (4.33a)$$

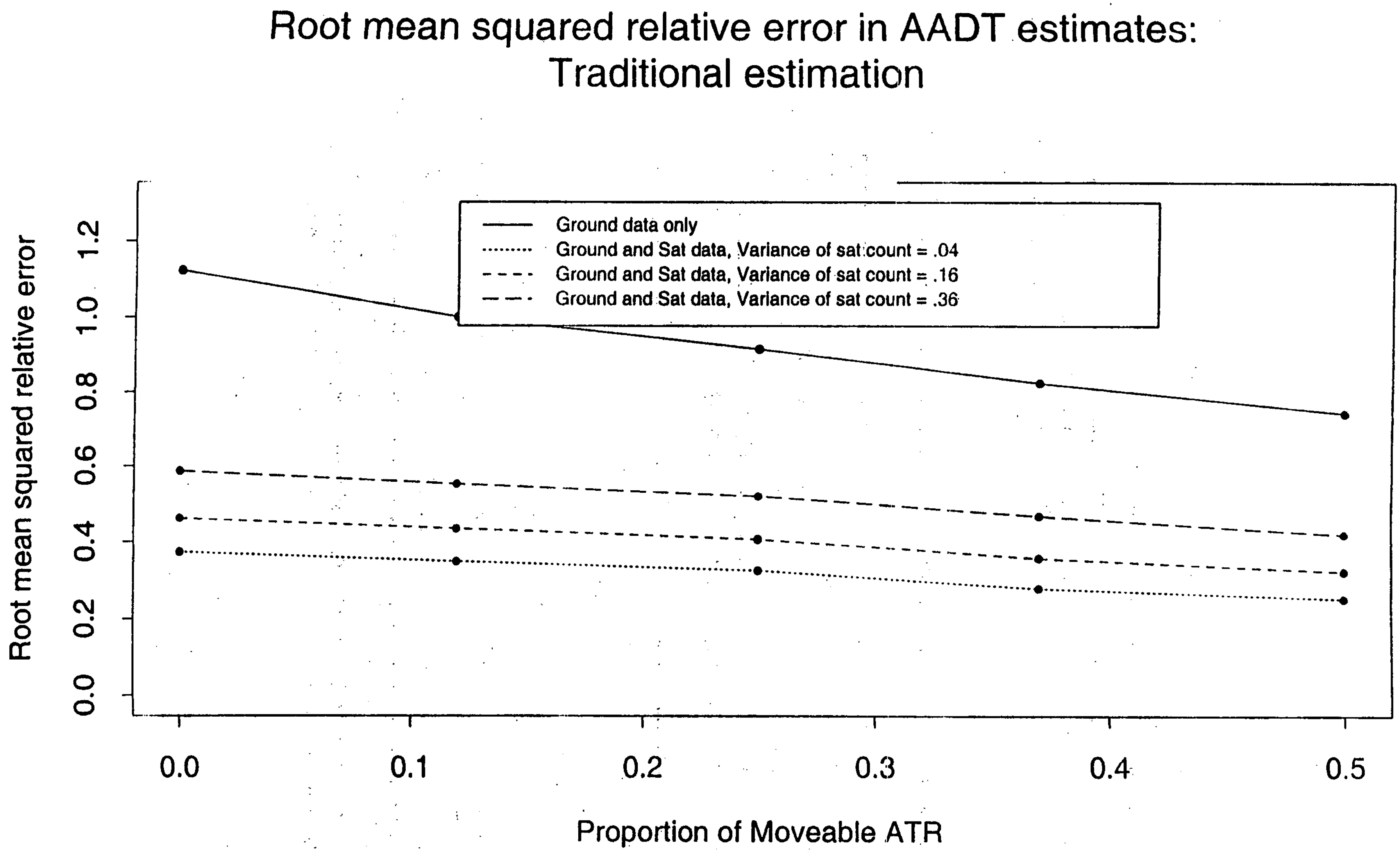
$$AREvmt^{()} = \Sigma_{r=1,\ldots,100}\ REvmt^{()}_{r}\ /\ 100\ , \qquad () = {}^{(g)}, {}^{(sg)}. \tag{4.33b}$$

In Figures 4.1a-4.1c, we graph the average AADT errors *ARMSREaadt* of Equation (4.32b) as a function of the number of movable ground sensors M when using only ground-based data (solid curve) and when combining satellite-based and ground-based data (dashed curves). In these figures, the abscissa portrays the number of moveable ground sensors as a proportion of the N=100 links in the network. That is, an abscissa value of 0.2, for example, is obtained from M/N = 20/100. The results in these figures were produced using the log-normal generation and traditional estimation programs. We present results for equivalent satellite coverage *ESC* approximately equal to 0.5 satellites (*i.e.*, $N^{l} = 5$) in Figure 4.1a, 1.0 satellite (*i.e.*, $N^{l} = 10$) in Figure 4.1b, and 1.5 satellites (*i.e.*, $N^{l} = 15$) in Figure 4.1c.

The different curves for the combined satellite-based and ground-based data represent the use of different variances of the error term in the satellite-based information. As mentioned above, we held $\sigma^{2(g)} = 0.04$ for all runs. The lowest combined satellite-based and ground-based data curves in the figures were produced with $\sigma^{2(s)} = 0.04$, representing a case where the 24-volume estimates from the satellite data would be as good as those obtained from ground sensors. This would be an unrealistic case, but it serves as a lower bound on the combined satellite- and ground-data case. The middle and highest combined satellite-based and ground-based data curves were produced with $\sigma^{2(s)} = 0.16$ and $\sigma^{2(s)} = 0.36$, respectively. As mentioned above, an appropriate relation between $\sigma^{2(s)}$ and $\sigma^{2(g)}$ is unknown at this time, and dete g such a relation would require future research. Still, we note that when $\sigma^{2(s)} = 0.36$ the variance of the error term used in producing satellite-based estimates would be nine times that of the error term used in producing the ground-based volumes, which could be considered a large increase.

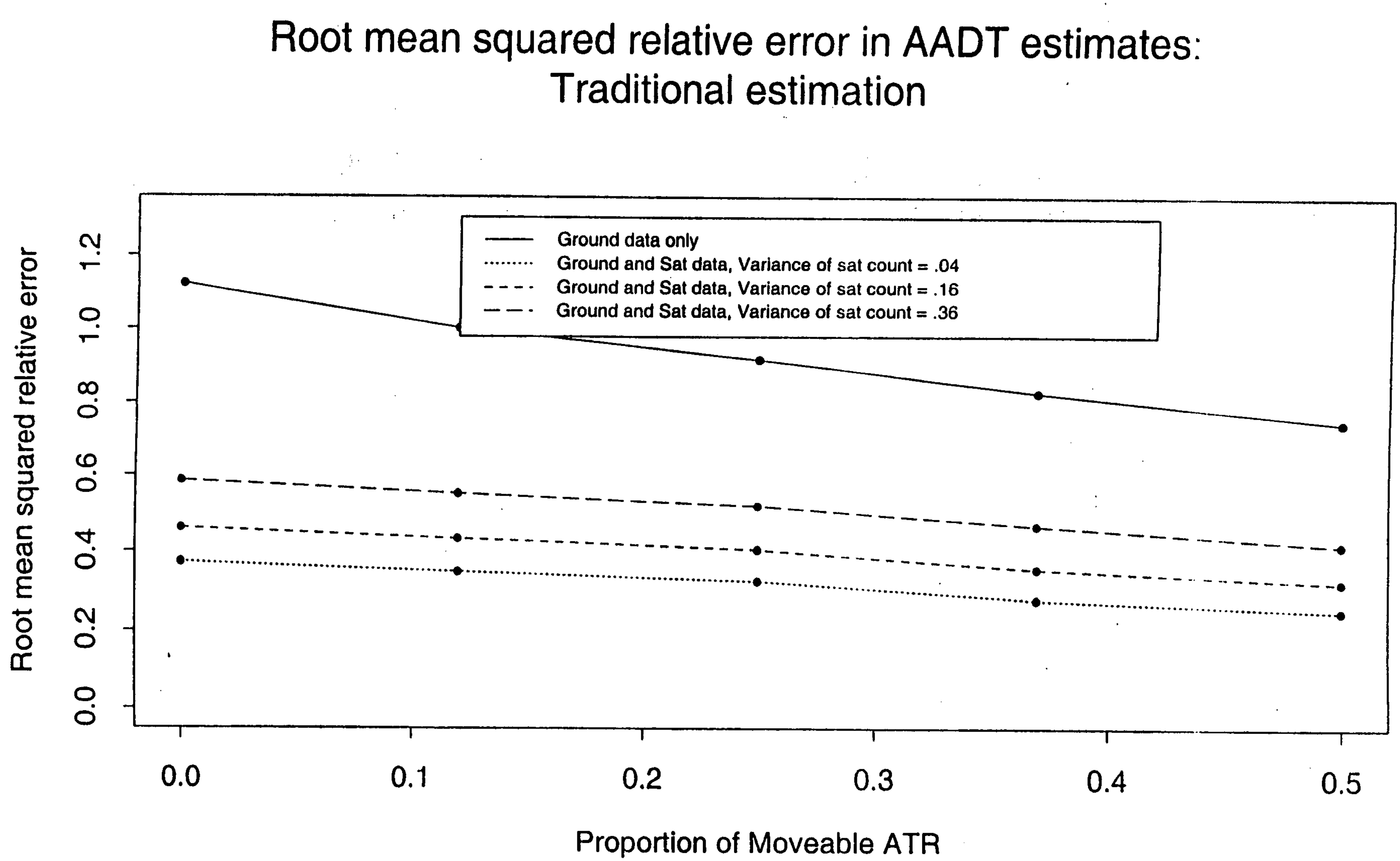
In Figures 4.1a-4.1c, we see that all the curves produced when combining satellite-based and ground-based data lie entirely below the curve produced when using only ground-based data. (The ground-based data only curve is the same in the three figures, since the figures differ only in the amount of equivalent of satellite coverage.) More specifically, even when covering up to 50% of the links per year with movable sensor (Proportion of movable ATR's – *0.50)* and when using the equivalent of only one-half of available satellite data (Figure 4.1a), using satellite data markedly decreases AADT error from that produced when using ground-based data only, even when the error associated with scaling up the satellite snapshot to a 24-hour volume is considered high ($\sigma^{2(s)} = 0.36$).

Note also that the error associated with using only ground-based data when the proportion of movable ground sensors is 0.50 (50% of the N links) is greater than that associated with combined satellite-based and ground-based data when the proportion of moveable ground sensors is 0.12 (12% of the N links), even in the $\sigma^{2(s)} = 0.36$ case and when using only half the available satellite data (Figure 4.1a). Since we are considering a time period of one year, a 0.12 proportion of movable ground sensors represents a scenario in which all the links of the network would be covered with movable counts



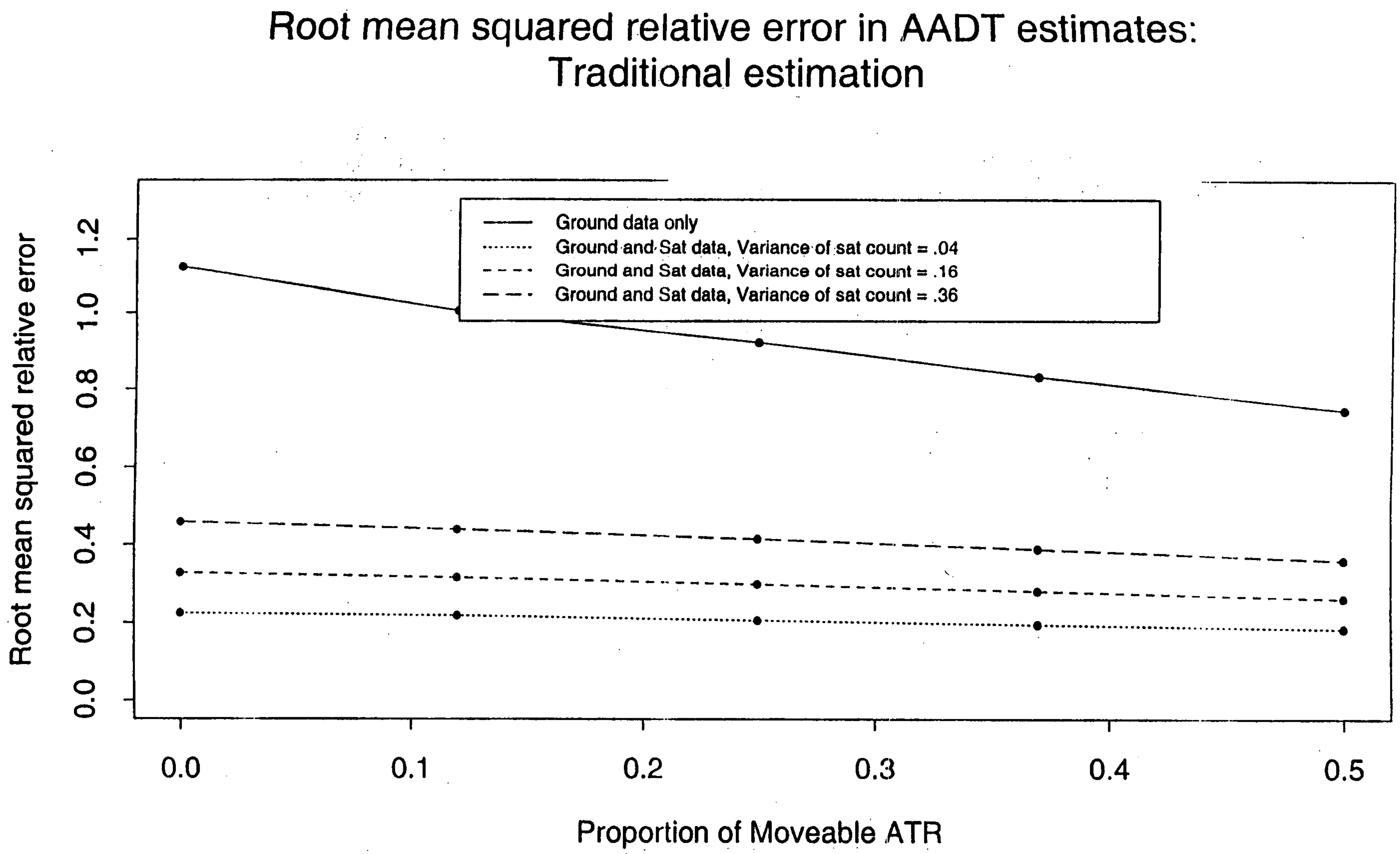
a. Equivalent Satellite Coverage *ESC* = 0.5.

Figure 4.1. Average root-mean-squared relative errors in AADT – *ARMSREaadt* – as a function of proportion of movable ground sensors and variance in satellite-based estimates $\sigma^{2(s)}$ when using only ground-based data and when combining satellite-based and ground-based data (log-normal generation; traditional estimation method).



b. Equivalent Satellite Coverage $ESC = 1.0$

Figure 4.1 Average root-mean-squared relative errors in AADT – *ARMSREaadt* – as a function of proportion of movable ground sensors and variance in satellite-based estimates $\sigma^{2(s)}$ when using only ground-based data and when combining satellite-based and ground-based data (log-normal generation; traditional estimation method).



c. Equivalent Satellite Coverage $ESC = 1.5$.

Figure 4.1 Average root-mean-squared relative errors in AADT – *ARMSREaadt* – as a function of proportion of movable ground sensors and variance in satellite-based estimates $\sigma^{2(s)}$ when using only ground-based data and when combining satellite-based and ground-based data (log-normal generation; traditional estimation method).

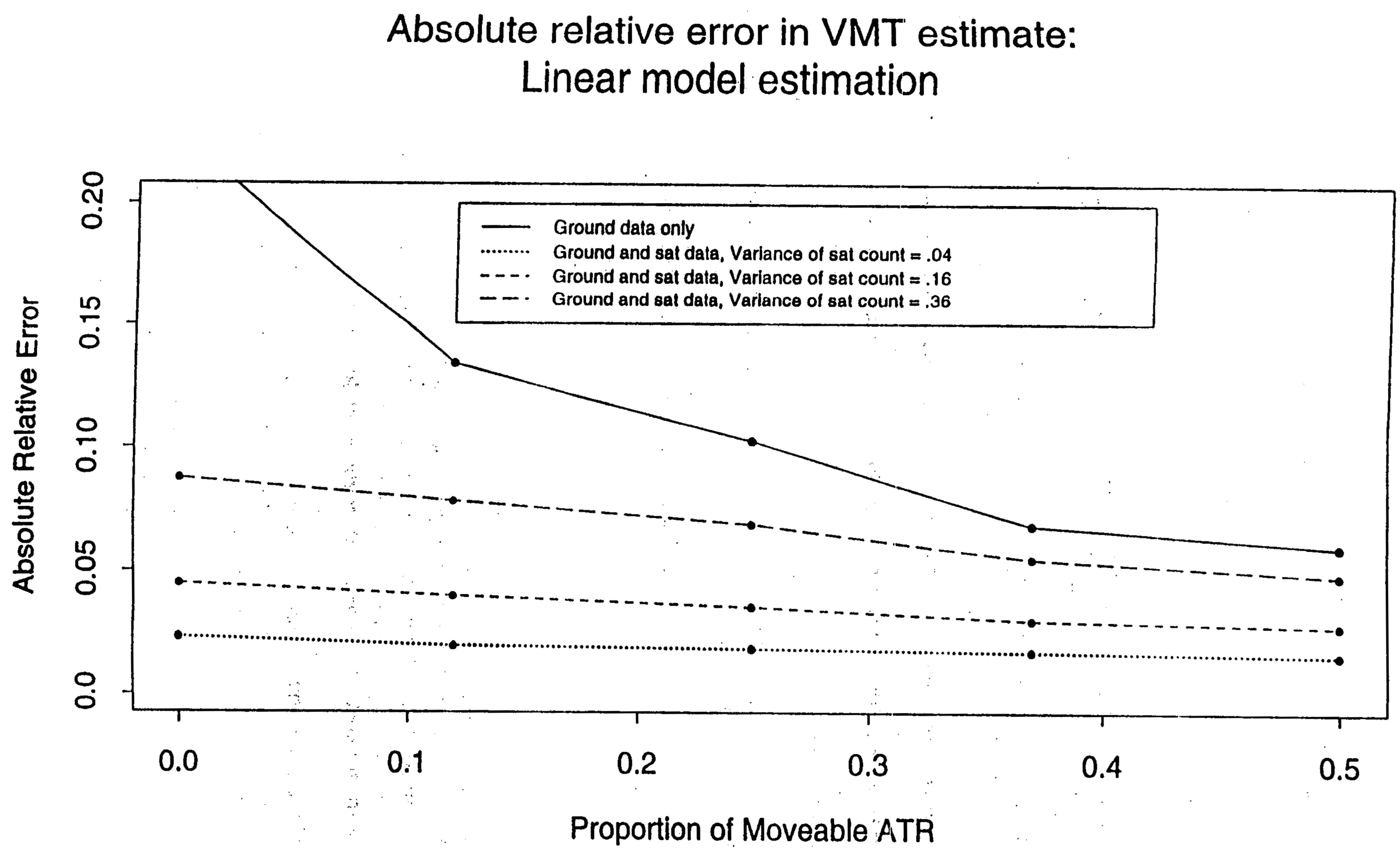
approximately every eight years, whereas a 0.50 proportion represents one in which the network would be completely covered with movable counts every two years. According to these results, then, incorporating satellite data into the estimation of AADT's would allow ground crews to operate on an 8-year cycle and still produce better estimates than if they operated on a 2-year cycle without satellite data, even when there is great variability in scaling up satellite snapshots to 24-hour volume estimates. Fewer DOT resources would be required for an 8-year cycle (*i.e.*, the "with satellite data" scenario) than for a 2-year cycle (*i.e.*, the "without satellite data" scenario).

In Figure 4.2 we graph the average relative VMT errors *AREvmt*'s of Equation (4.33b) for equivalent satellite coverage of 1.0. Again, we see that the combined satellite-based and ground-based data curves lie below the ground-based only data curve. From the figure, we see again that the error when a proportion of 0.12 of the links is covered with movable ground sensors on the $\sigma^{2(s)}=0.36$ combined satellite-based and ground-based data curve is no worse than the error at a 0.50 proportion on the ground-based data only curve. That is, covering the links of the network with movable ground sensors on an 8-year cycle when incorporating satellite data would lead to VMT estimates that are as accurate on average as those produced when covering the network on a 2-year cycle when not using the satellite data, even when scaling up satellite snapshots to 24-hour estimates is very "noisy" (high $\sigma^{2(s)}$).

In Figure 4.3 we graph the average VMT errors *ARMSREvmt* of Equation (4.33b) when using the traditional estimation method, but when assuming that volumes are generated from a Poisson distribution. We again graph as a function of the number of movable ground sensors *M* when using only ground-based data (solid curve) and when combining satellite-based and ground-based data (dashed curve). Since there is only one parameter of the Poisson distribution (the mean), we cannot parameterize the simulation by variances, as in the log-normal case. Therefore, there is only one curve for the combined satellite-based and ground-based data estimation.

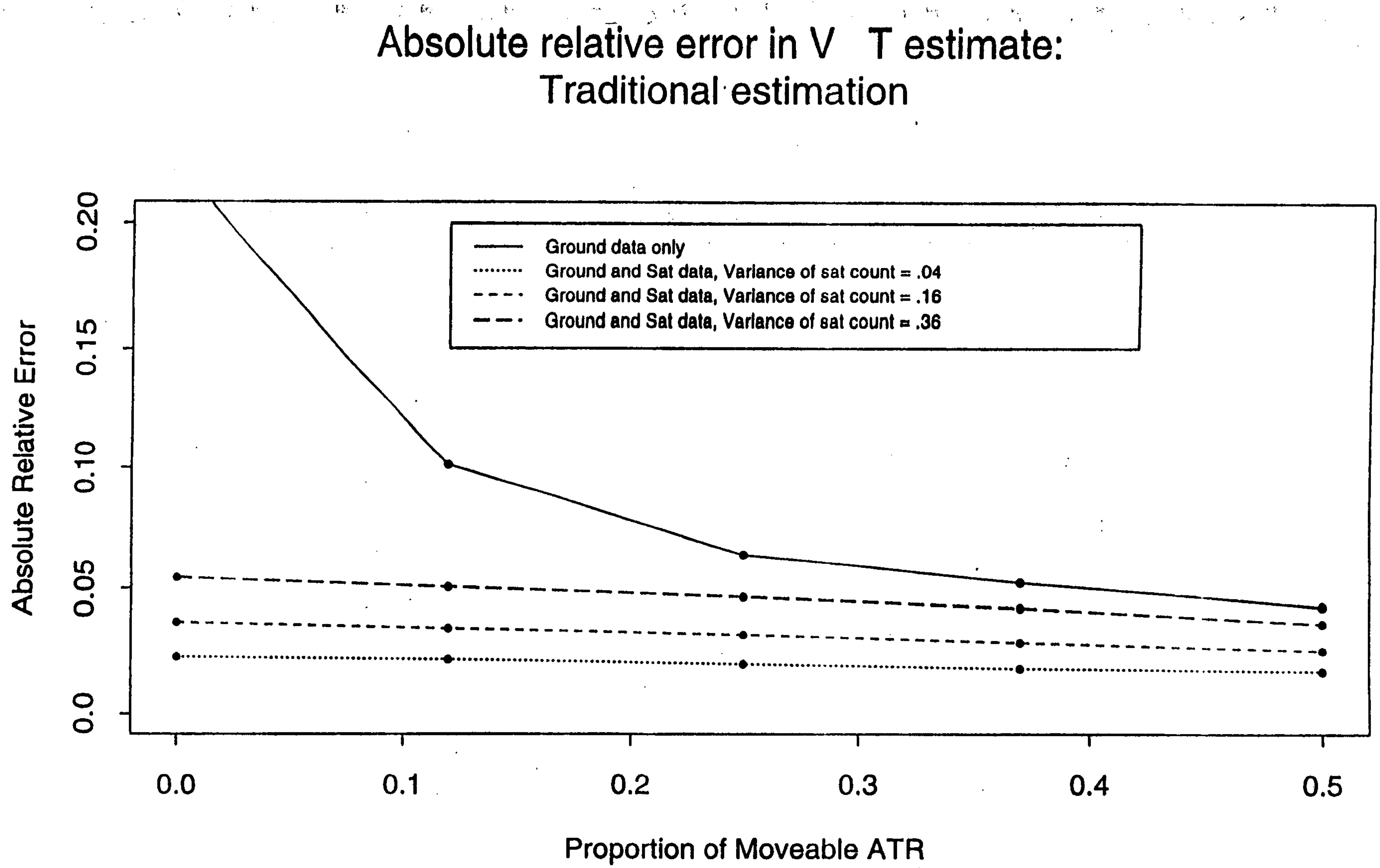
Under this different set of assumptions (Poisson generation) the value of the satellite data in reducing the error in AADT estimation is again strikingly apparent. The curve produced when combining the satellite-based data with the ground-based data lies below that produced when using only the ground-based data. Again, covering the network with ground-based counts on an 8-year cycle when coupled with satellite data produced better results than covering the network on a 2-year cycle without satellite data.

We also investigated the improvements that would stem from using the model-based estimation procedure with the log-normal generation assumption (see Section 4.1.1.1). As we did in Figures 4.1a-4.1c, we plot in Figures 4.4a-4.4c the average AADT errors *ARMSREaadt* of Equation (4.32b) as a function of the proportion of movable ground sensors when using only ground-based data (solid curve) and when combining satellite-based and ground-based data (dashed curves). Whereas the results in Figure 4.1a–4.1c were produced when using the traditional estimation method, the results graphed in Figures 4.4a-4.4c were produced when using the model-based method.



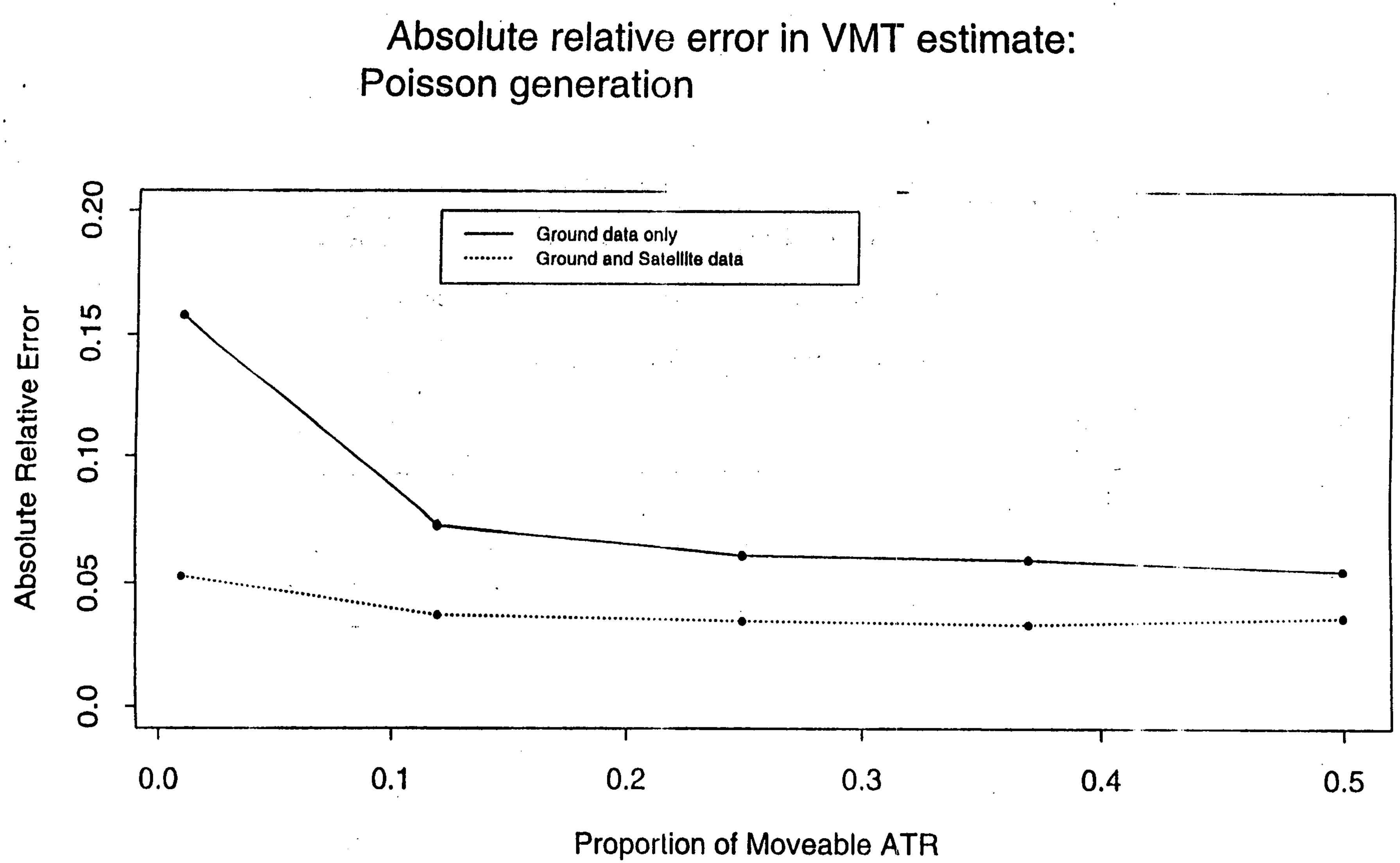
a. Linear model generation; equivalent satellite coverage *ESC* = 1.0.

Figure 4.2 Average relative errors in VMT *AREvmt* as a function of proportion of movable ground sensors and variance in satellite-based estimates $\sigma^{2(s)}$ when using only ground-based data and when combining satellite-based and ground-based data (linear model generation; traditional estimation method).



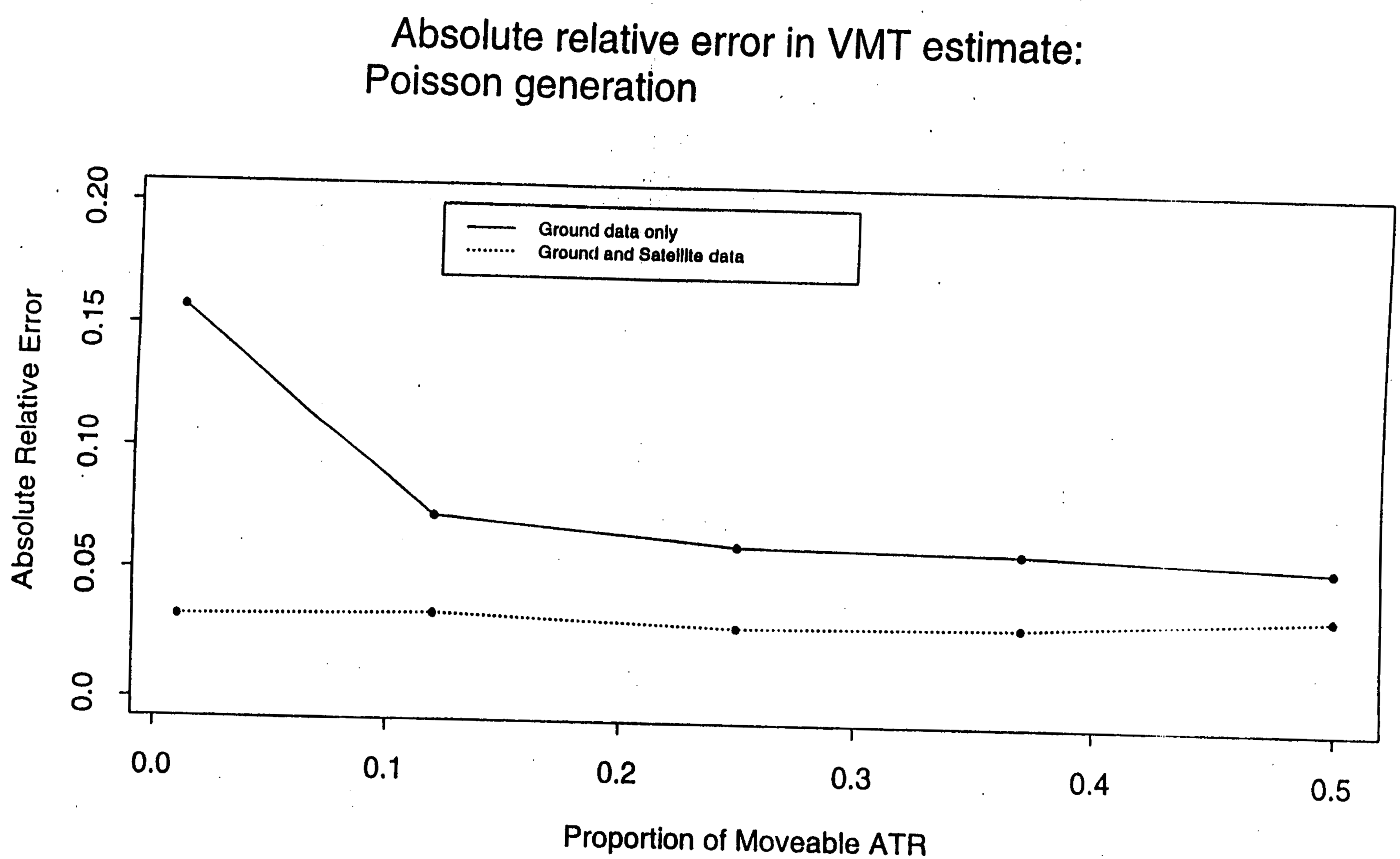
b. Traditional estimation method; equivalent satellite coverage $ESC = 1.0$.

Figure 4.2 Average relative errors in VMT *AREvmt* as a function of proportion of movable ground sensors and variance in satellite-based estimates $\sigma^{2(s)}$ when using only ground-based data and when combining satellite-based and ground-based data (linear model generation; traditional estimation method).



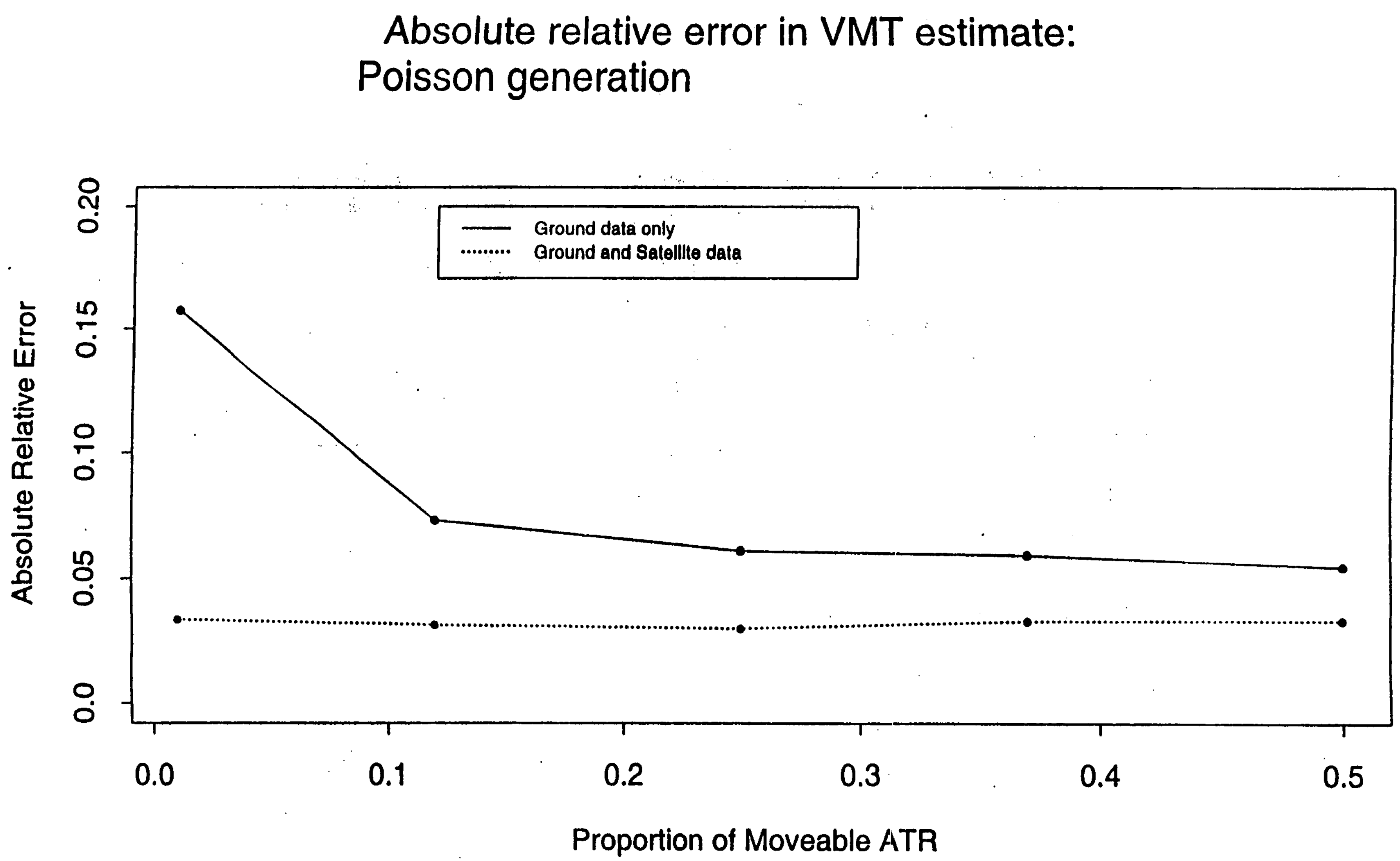
a. Poisson generation; equivalent satellite coverage $ESC = 0.5$.

Figure 4.3 Average root mean squared relative errors in VMT *AREvmt* as a function of proportion of movable ground sensors when using only ground-based data and when combining satellite-based and ground-based data (Poisson generation; traditional estimation method; equivalent satellite coverage $ESC = 1.0$).



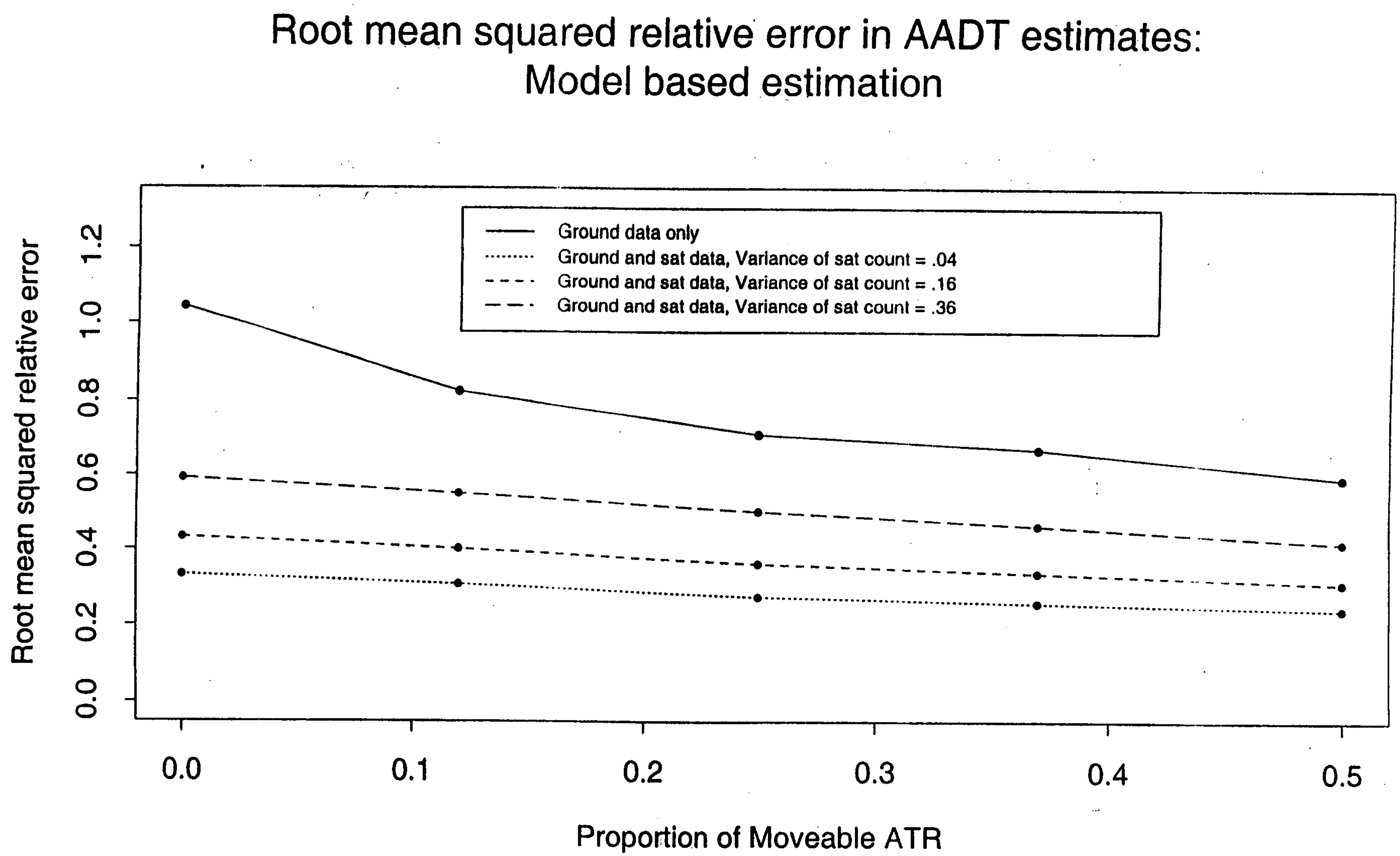
b. Poisson generation; equivalent satellite coverage $ESC = 1.0$.

Figure 4.3 Average root mean squared relative errors in VMT *AREvmt* as a function of proportion of movable ground sensors when using only ground-based data and when combining satellite-based and ground-based data (Poisson generation; traditional estimation method.



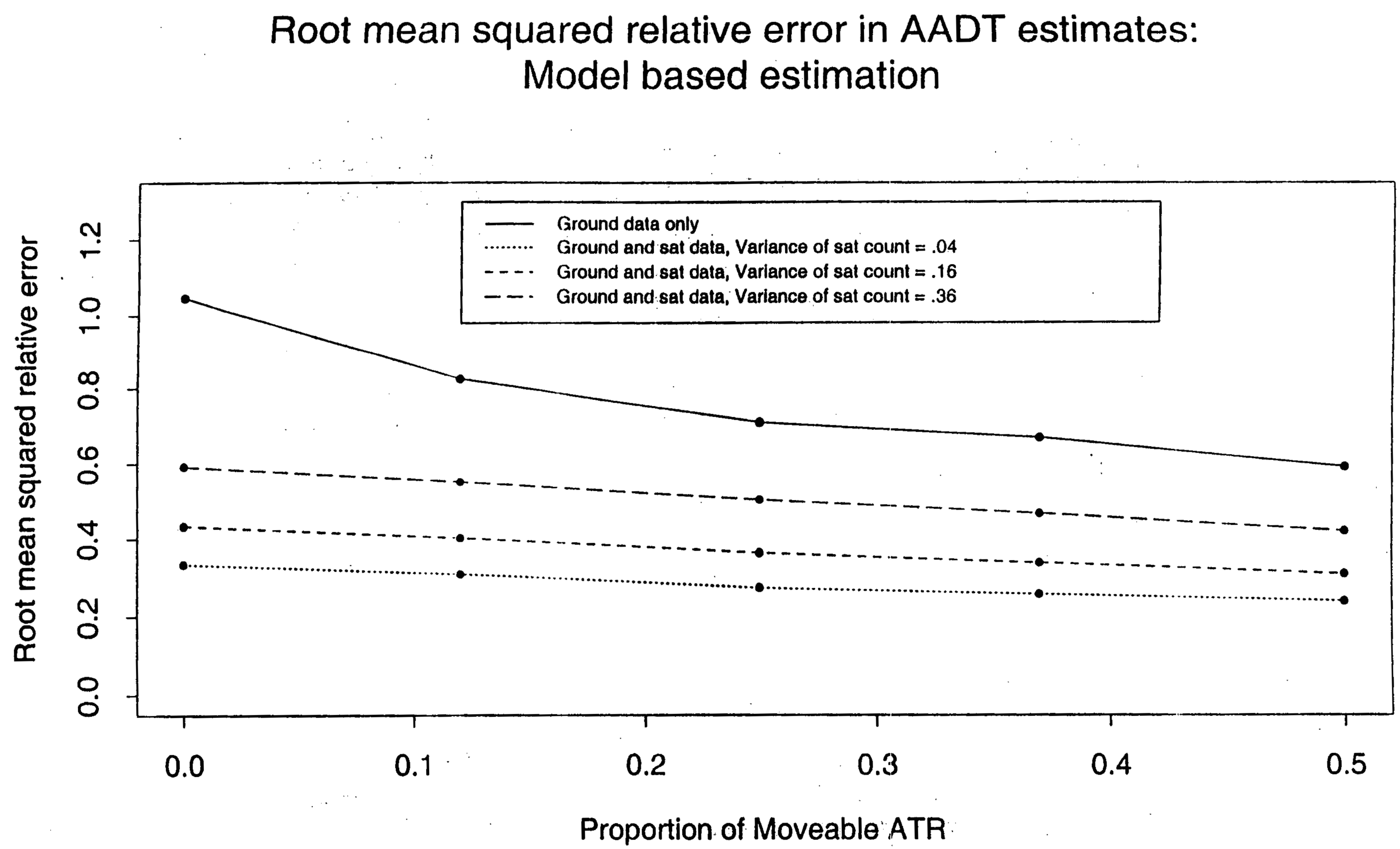
c. Poisson generation; equivalent satellite coverage $ESC = 1.5$.

Figure 4.3 Average root mean squared relative errors in VMT *AREvmt* as a function of proportion of movable ground sensors when using only ground-based data and when combining satellite-based and ground-based data (Poisson generation; traditional estimation method).



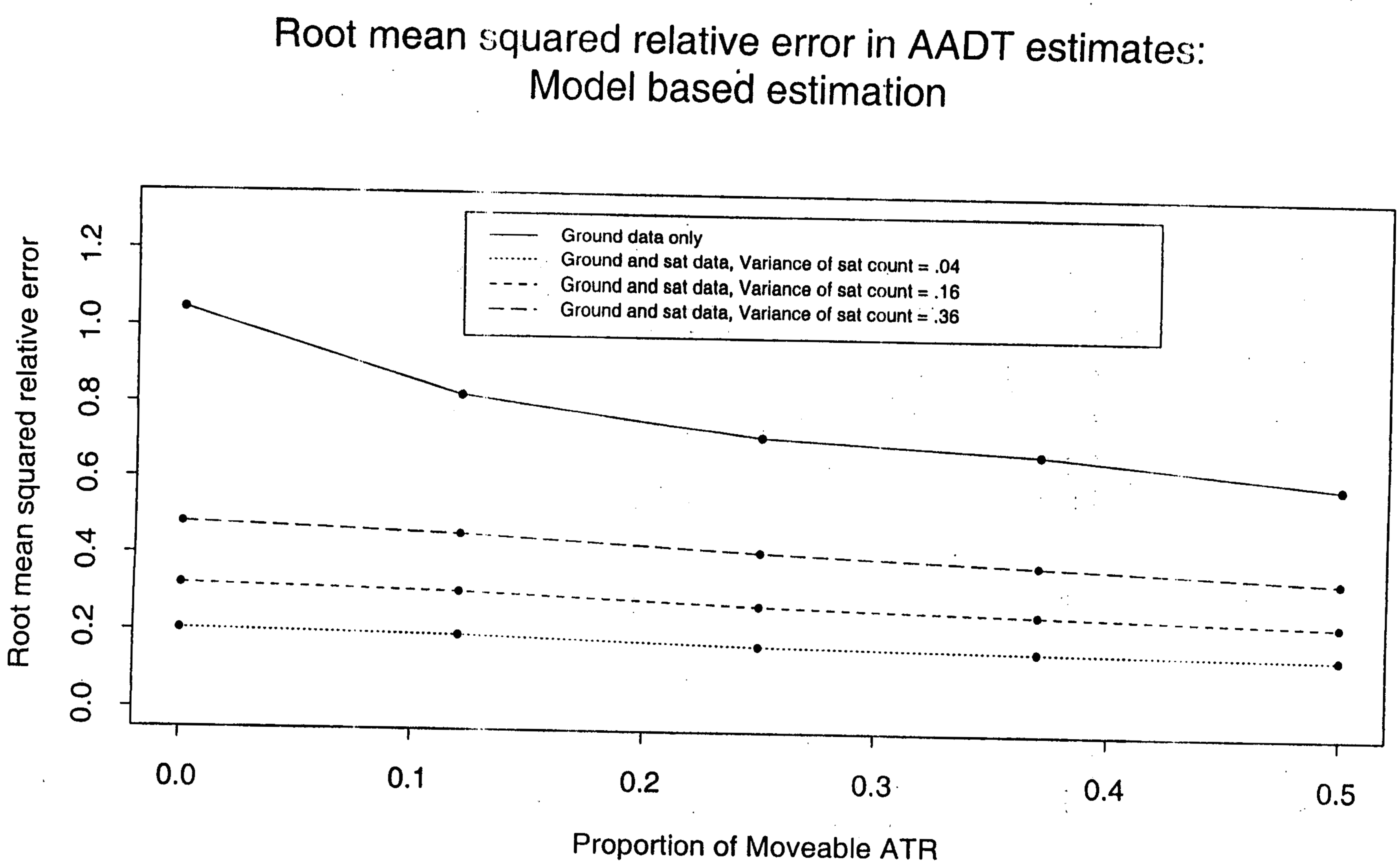
a. Equivalent satellite coverage $ESC = 0.5$.

Figure 4.4 Average root mean squared relative errors in AADT *ARMSREaadt* as a function of number of proportion of ground sensors and variance in satellite-based estimates $\sigma^{2(s)}$ when using only ground-based data and when combining satellite-based and ground-based data (log-normal generation; model-based estimation method).



b. Equivalent satellite coverage $ESC = 1.0$.

Figure 4.4 Average root mean squared relative errors in AADT *ARMSREaadt* as a function of number of proportion of ground sensors and variance in satellite-based estimates $\sigma^{2(s)}$ when using only ground-based data and when combining satellite-based and ground-based data (log-normal generation; model-based estimation method).



c. Equivalent satellite coverage $ESC = 1.5$.

Figure 4.4 Average root mean squared relative errors in AADT *ARMSREaadt* as a function of number of proportion of ground sensors and variance in satellite-based estimates $\sigma^{2(s)}$ when using only ground-based data and when combining satellite-based and ground-based data (log-normal generation; model-based estimation method).

The results in this set of figures again show that adding satellite data is markedly reduces estimation errors even in the high variance ($\sigma^{2(s)}=0.36$) case and when using only half the satellite data ($ESC - 0.5$). Once again, lower average error is produced from covering the network with ground counts on an 8-year cycle with only half the satellite data ($ESC = 0.5$) than on a 2-year cycle without satellite data, even in the high variance case (see Fig. 4.4a).

A comparison of the Figures 4.4a–4.4c curves to their counterparts in Figures 4.1a-4.1c shows that our model-based estimation method improved on the traditional estimation method. The improvement was most pronounced when using ground-based data only and seemed least pronounced for the combined satellite-based and ground-based data curves with high variance in the satellite error term ($\sigma^{2(s)}=0.36$) with the highest satellite coverage ($ESC=1.5$).

We also note that the $\sigma^{2(s)}-0.36$ combined satellite-based and ground-based data curve produced when using the traditional estimation method has smaller errors than the ground-based only data curve produced when using the model-based estimation method. That is, even when the satellite-based data are "noisy," using these noisy data with an inferior (traditional) estimation method decreases AADT estimation errors more than using a better (model-based) estimation method without the data,.

The errors graphed in Figures 4.4a-4.4c are based on averages over 100 replications of a one-year analysis period. In Appendix F, we present scatter plots of the 100 paired (traditional method *vs.* model-based method) *RMSREaadt* values of Equation (4.32a) when using only ground-based data and when combining the satellite-based and ground-based data at various $\sigma^{2(s)}$ values for 0.25 (M=25) and 0.50 (M=50) proportion of links covered with moveable ground-based sensors, and at $ESC=1.5$. Comparing the results of Figures 4.4a-4.4c to Figures 4.1a-4.1c, we saw that our model-based estimation method performs better on average than the traditional method. The scatter plots confirm that the model-based method does better than the traditional method in most individual replications. Still, there are many cases where the traditional method outperforms the model-based method, and we feel that future improvements could be made to our model-based method.

Section 5. Summary and Future Work

In this report, we documented progress on three issues that would need to be addressed before high-resolution satellite imagery could be used to complement traffic monitoring programs:

- demonstrating that vehicles can be identified and classified accurately from real satellite imagery;
- developing efficient image processing methods;
- dete g methods to integrate the imagery with ground-based data and assessing the value of this integration.

Although substantial progress has been made, we feel that further work is needed in each of these areas.

We have been developing a methodology to compare vehicle classifications obtained from satellite images with those obtained from traditional ground counts and writing software that would automate much of the analysis. The results of field tests designed to demonstrate the methodology, where we used scanned aerial photographs to simulate satellite imagery, were encouraging and instructive.

When high-resolution satellite data becomes available, the methodology we have been developing should be applied to show that vehicles could in fact be identified and classified in high-resolution satellite imagery. Because of the different types of data – data obtained over space at an instant of time in the images, and data obtained over time at a point in space in the ground data – discrepancies can occur between the two classifications. These discrepancies can occur even if every vehicle is correctly identified and classified in the satellite imagery. Therefore, we suggest that more work be devoted to reducing the size of this discrepancy and developing a maximum size of discrepancy that can be tolerated and still conclude that vehicles are being classified acceptably in the two data sets. When pl g for tests with real satellite data, additional thought will also have to be given to differences that can arise when using real satellite data. For example, thought should be given to differences in data format, the ease with which the appropriate highway segments can be identified in large area images, and an edge detection algorithm to efficiently determine the highway edge lines.

We are also encouraged by the progress made in our image processing approach. Specifically, we have developed a means to transform the steady-state *background* image of a highway segment to those of a *time t-image* that is to be analyzed for vehicles. Our objective is to classify the subtracted pixel values of the two images into dynamic and static pixels, where the dynamic pixels would serve as an indication of movement attributed to vehicles. Experiments on simulated images and scanned aerial photographs produced encouraging results and demonstrated the robustness of the results to prior estimates of traffic density, estimates required as input to our approach.

Future work would be necessary to develop, test, refine, and code the image processing algorithms we have been developing. Until now, we have used simulated images or scanned aerial photographs to serve as the steady-state background images of the highway pavement. In practice, we would expect that the background image would be constructed from a series of images taken over time. For example, the background image could be obtained by averaging images of a specific segment acquired at different times. Each time a new image is acquired, it would be combined with the present background to form an updated background image. Averaging the images should substantially reduce the contribution of the dynamic signals (principally, vehicles) after a sufficient number of observations, leaving a background image that corresponds almost entirely to an average of pavement signals. This averaging procedure could be tested using a series of satellite images when such images become available. Until then, a series of scanned aerial photographs or digital photographs of the same highway segment at different times could be used. This approach is motivated by an assumption that the dynamic (vehicle) signals are sufficiently few that they would be filtered out after averaging a few images. This should be the case on lower vehicle density highways. However, it is also necessary to determine a good procedure for constructing the background image on highways with higher vehicle densities.

It appears that our transformation procedure is working well. Still, it should be tested more systematically and under a variety of conditions. It would be more efficient to conduct large-scale testing on simulated images, but some real images – either scanned aerial photographs, digital images taken from an aircraft, or real satellite images – should also be investigated to ensure reasonableness of the process generating the simulated images.

The transformation and subtraction procedure must also be integrated with a vehicle classification module. The classification module would operate on the pixels that received a sufficiently high probability of being dynamic after subtracting the transformed background image from the time *t*-image. Decision rules can be used to determine whether groups of such dynamic pixels constitute a vehicle or a nonvehicle object. If the group of pixels is identified as a vehicle, the group of pixels must then be classified by vehicle type. Previously, we developed rules to operate on a binary output of a thresholding procedure (Merry *et al.* 1996). These rules worked well in conditions where vehicle shadows were pronounced. We feel that it will be possible to modify these rules to work well with our transformation and subtraction approach under a wider set of conditions, but other methods should also be investigated.

Further work is also warranted in determining the value that imagery data would add to traffic monitoring programs and to integrating these data with those obtained from ground sensors. We have been concentrating on estimating Average Annual Daily Traffic (AADT) and Vehicle Miles Traveled (VMT). Based on results produced from the simulation and estimation programs we have developed, it appears that adding satellite data to ground-based data would improve the quality of the AADT and VMT estimates while requiring fewer ground personnel to collect ground-based traffic counts.

These encouraging results were obtained even when using methods similar to those traditionally employed, methods that were not designed to take advantage of the two different types of data. Our first attempt at "model-based" methods improved the estimates further. However, we expected to see greater improvement with the model-based method, and we therefore feel that this method can be refined in the future. Also, the method should be investigated for robustness to data that are not entirely compatible with the assumed model. More radically different methods should also be investigated for combining ground-based and image-based data more effectively – for example, methods that take advantage of spatial correlation in the traffic patterns that can be observed in the satellite images.

We also feel that slight modifications in the generation and estimation software we have developed would produce powerful tools for investigating other questions. For example, this type of software could be used to identify temporal patterns in traffic flows that lead to especially large or small additional value that could be contributed by the satellite data. Such knowledge would ultimately be useful in deciding which highway segments to target with pointable satellite sensors. The software could also be used to assess the relative effectiveness of ground-based sampling patterns when using satellite data. This information could then be used to design sampling strategies in state Departments of Transportation (DOT's), or other agencies interested in estimating AADT and VMT.

In addition, other issues not addressed in this study should be investigated if satellite imagery is to be incorporated in traffic monitoring or other transportation programs. For example, institutional issues associated with obtaining data in standard formats on a long-term and reliable basis, preprocessing these data, making them accessible to state DOT's, and having the DOT's integrate them into their operations would need to be addressed. Moreover, exploring the use of the imagery data to identify parameters other than AADT or VMT seems ted. For example, the image data could be useful in developing classes for volume or weight samples, targeting resources for speed studies, detecting high truck volumes on alternative routes to those passing open weigh stations, or calibrating flow prediction models.

References Cited

American Society of Photogrammetry and Remote Sensing, 1996. "Land satellite information in the next decade – The World Under a Microscope," Executive Summary, American Society of Photogrammetry and Remote Sensing, Bethesda, Maryland, 72 p.

Castelman, K.R., 1996. *Digital Image Processing,* Prentice Hall: Upper Saddle River, New Jersey.

Chambers, J.M., and J. Trevor J. Hastie [eds.], 1992. *Statistical Models,* S. Wadsworth and Brooks/Cole, Pacific Grove, California.

McCord, M.R., C.J. Merry and J.D. Bossler, 1995a. The feasibility of traffic data collection using satellite imagery, Final Report to Federal Highway Administration, The Ohio State University, Research Foundation: Columbus, Ohio, April, 208 p.

McCord, M.R., C.J. Merry, X.D. Sun and F. Jafar, 1995b. Resolution effects on vehicle counts and classification through remote sensing, *Journal of the Transportation Research Forum,* 35(1):41-52

McCord, M.R., C.J. Merry and P. Goel, 1998. Incorporating satellite imagery in traffic monitoring programs, Proceedings of the North American Travel Monitoring Exhibition and Conference (NATMEC '98), Charlotte, North Carolina, 11-15 May, 18 p.

McShane, W.R. and R.P. Roess, 1990. *Traffic Engineering,* Prentice-Hall: Englewood Cliffs, New Jersey, 660 p.

Merry, C.J., M.R. McCord, J.D. Bossler, F. Jafar, L.A. Peréz, 1996. Feasibility of using simulated satellite data coordinated with traffic ground counts, Final Report to Federal Highway Administration, The Ohio State University, Research Foundation: Columbus, Ohio, August, 80 p.

U.S. Department of Transportation, 1992. Traffic Monitoring Guide, U.S. Department of Transportation, Federal Highway Administration, Office of Highway Information Management, FHWA-PL-92-017, October, Washington, DC.

Appendix A. Description of the Software Code for Computing Traffic Measures

Introduction

We developed software that computes traffic measures at a location on the highway during a time interval from a snapshot of the highway. We called this software COUNT. The basic input data for COUNT are the highway axis, vehicle records, count location, and the time interval during which the measures are to be computed. The output data are traffic measures during this time interval at the specified location. Additionally, this software has the capability to compute the maximum time interval allowed by the highway limits for extracting traffic measures. COUNT is written in FORTRAN and is complied and linked using a FORTRAN-77 compiler on a workstation platform. It can easily be adapted to any other FORTRAN compiler or other platforms.

In this chapter we describe the input data required by this software, the output, and the code of the software. The next section describes the input and output data and gives examples of the data format. The following sections describe the various modules of the program.

Software Input Data

In this section we describe the input data for COUNT and provide examples to illustrate these data. The data format described is that read by the version of COUNT used at the time of this writing. This version is the one described in here. All the components of the data must be included as input to COUNT; however, the format and order of the components can be changed. The modules that read the input data may be modified to read the input in different formats. Thus, the input format would have to be changed to fit the requested input format by that version.

In this section, we first explain the highway axis data, then the vehicle record data, the highway count data, and the highway limit data.

Highway Axis Data

Highway axis data are used as an axial reference for all the vehicle locations on the highway at different times. The Euclidean distance computed using the coordinates of two locations would determines the straight line distance between these two locations on the highway. However, distances on highways are not necessarily straight. For example, a vehicle does not travel in a straight line when navigating a horizontal curve. The highway geometry can be represented by the highway axis. The axis is a linear feature of the highway. We found it useful to have this axis correspond to the inner edge of road pavement. In this research, we refer to this highway inner edge axis as axis for simplicity.

Highway axis data used in this software are a highway datum point and the digitized highway axis coordinates. The datum point is an arbitrary distance corresponding to the first point of the axis. It could, for example, be the linear distance from a known landmark on the road to the point, the mile marker distance of the point, or any other arbitrary distance specified. The coordinates of each digitized point are denoted (xa_i,ya_i), where xa_i refers to the xa coordinate of the ith digitized point and ya_i refers to the ya coordinate of the ith point on the axis. These coordinates could be given with reference to any coordinate system, but the digitized axis coordinates for one highway segment should refer to the same coordinate system and datum.

This version of COUNT assumes that the datum point is given in units of meters because this software is set to process images with resolution given in Metric units. Figure A-1 shows an example of a highway axis input data file corresponding to the images shown in Figure A-2. The first line in this data is the datum point distance, which was arbitrarily set to a value of 2000. If desired, the real mile marker distance could have been used as the reference distance for the datum point. We choose the datum value to be some distance greater than zero so that if an extension beyond the beginning of the highway axis is extended by some distance from the starting end, the axis distance in the extended part of the axis will remain positive. We explain this aspect in more detail when we talk about the highway axis module.

Axis coordinate data start on the second line in Figure A-1. This line contains the coordinates of the datum point of the highway axis whose arbitrary distance was given in the first line. In this example, the point at xa = 1087 and ya = 6106 is 2000 m from some datum. The coordinates of the following points along the axis follow in order.

Vehicle Record Data

Location and time data for imaged vehicles are also required as input for this software. Using the location of a vehicle on an image, the time when the vehicle was imaged at that location, the location of the count point, and the speed of the vehicle, the time when the vehicle would pass the count point is estimated. (Count point is where vehicles are to be estimated to pass during the time interval of interest.) Using time and location records of a vehicle in two consecutive images, the average speed of the vehicle when traveling between these two locations can be estimated.

Each vehicle observed in an image receives a record. Records of vehicles in different vehicle classes are saved in separate lists. The version of COUNT described here only considers two classes of vehicles, large vehicles and small vehicles. For simplicity we refer to them as trucks and cars in this research. Thus, the vehicle records are sorted into two lists, one list for cars and one list for trucks. A record contains information that identifies the vehicle with an integer identity number, locates it in the coordinate system through its x and y coordinates, and indicates when the vehicle was at the given location with a time stamp.

A vehicle that is imaged more than once will have more than one record. However, the integer identification number would be the same for different records corresponding to this vehicle. Identifying the same vehicle at different locations in different images leads to velocity estimates of the vehicle. The velocity is estimated as the distance traveled between the image when the vehicle was at these locations.

Vehicle coordinates are the coordinates of the vehicles located with reference to the coordinate system used for the highway axis. The vehicle coordinates are referenced by (x^v_j, y^v_j), where x^v_j represents the x coordinate of the jth vehicle and y^v_j represents the y coordinate of the jth vehicle. The time when the vehicle is seen at the specified location (x^vj, y^v_j) is the time when the vehicle was imaged at these coordinates.

To illustrate, consider the vehicle record data in Figure A-3. The first line in the file is a 1 to indicate that the following are records of cars, which are identified as class 1 of vehicles in this study. The first line in the records of cars contains the record of a vehicle that is identified as car 4. The following two numbers are the x and y coordinates of the location of this car. The last number in the record is the time when this car 4 was at these coordinates, represented in hours:minutes:seconds. This line indicates that car 4 was at x=1018 and y = 5846 at time 10:54:31. The second line contains the records of vehicle 5. This record indicates that vehicle 5 was at x=1017 and y=5828 at 10:54:31. Line 7 contains the records of car 11, which indicates that car 11 was at x=921 and y=5451 at 10:54:31. Lines 17 and 24 also contain records of car 11. However, these records correspond to car 11 being imaged at times 10:54:36 and 10:54:41, respectively. Line 27 has the values (-1, -1, -1, -1, -1, -1). This is the indictor for the end of car data. The next line contains a 2, which indicates that the following data are data records of trucks, the record category of vehicles in this study. The truck data are arranged in the same format as the car data. Like the car data, the last line has the values (-1, -1, -1, -1, -1, -1), which indicate the end of the data in this class. If more vehicle classes are eventually used, then class numbers can be added. The module that reads the data would have to be modified to read data of more classes. We will indicate the lines code where this module needs to be modified to read more data when we explain the modules in the following sections.

Vehicle data are listed in order of the time when the images were taken. The records of the vehicles imaged at an earlier time are listed before the records of the vehicle imaged at a later time. The software assumes that the data are arranged in this time ascending format in the input data file. If vehicles are not arranged in an ascending order, we could write a module to rearrange it in this ascending format.

Highway Count Data
To compute level of service measures at a location, the software requires highway parameters, count location data, and count interval: Highway parameters are the number of lanes of the highway and the passenger car equivalent of trucks. The number of lanes of the highway must be recorded for the specific highway at the given location. The passenger car equivalent of a truck is also predefined for the specific highway depending on the terrain of the highway at the specific location. (Highway terrain is classified as level, rolling, or mountainous, and each type of terrain has a different passenger car equivalent of trucks for different highway class (see Highway Capacity Manual (TRB, 1997).) Count location data consist of the x and y coordinates at the location on the highway where the traffic measures are estimated. (Traffic measures are estimated at a point location on the highway to compare the measures estimated from the image data to the measures estimated from at ATR location at this point. This work was motivated in a large part by our desire to compare measures estimated from satellite data to those estimated from ATR data.) The time interval is the time during which traffic measures are computed at the count location. We denote the beginning of this time interval by t^1 and the end by t^2.

To illustrate, consider the example of highway count data in Figure A-4. These data correspond to the same highway for which the axis and vehicle data in Figures A-1 and A-3 were obtained. The highway has three lanes (line 1) and has a passenger car equivalent of trucks of 1.5 (line 2) (The passenger car equivalent of 1.5 was obtained from Table A-1 of the HCM for level terrain. The three lanes and 1.5 passenger car equivalent are entered to this input file manually.) The count location coordinates are (x=903, y=5393) and the time interval for the count begins at10:54:30 (line 5) and ends at 10:55:00 (line 6).

Highway Location and Limit Data
We mentioned earlier that the COUNT software has the capability to compute the largest time interval allowed by the highway limits for extracting measures. Given images of a

highway segment we can estimate traffic measures at any location on this highway. Time interval for computing these traffic measures is limited by the length of the highway segment imaged or by ramps. This software requires the limits of the highway and the count location as an input to compute the largest possible time interval for computing traffic measures. The highway location is defined by the x and y coordinates of the count location. Highway limit data include the farthest points of the highway that have been imaged. Figure A-5 shows an example of count location and highway limit data. The first two lines present the coordinates (x=1011, y=5795) of the count location. The next two lines indicate the coordinates of the limits of the highway. For example, the first limit of the highway is at (x=1080, y = 6077) and the other limit is at (x=947, y= 5530).

SOFTW MODULES

In this section we describe the main program of the COUNT software and its various modules. We present the general logic in flowcharts and explain the code in detail. COUNT first reads the highway axis from input files described in the previous section and computes the linear distances of these points from the datum. It then reads the vehicle coordinate data from input files and projects the vehicle coordinates to locations along the highway axis defined by the highway axis coordinates. Then the software gives the user the option to compute traffic measures during a specified time interval at a specified location, or to compute the largest time interval possible for computing traffic measures at a specified location for given highway limits. If the user chooses to compute traffic measures during a specified time interval, the software requires the user to input the count location and count time interval. If the user asks the software to compute the time interval, the software requires the user to input the count location and the highway limits. Figure A-6 shows the general flowchart of this software.

The Main Program

The main program declares variables and calls modules. This program is listed in Appendix A1. Lines 4 through 63 in this listing declare the variables used in the program. Comment lines have been added to explain where each variable is first used in the program.

The main program first calls the module CENTERLINE. This module reads the highway axis data and computes the axial distances from the original data of the coordinates in the highway axis data file. The command to call this module is in line 66 of the main program listing found in Appendix A1. In line 67, the main program then calls MINMAX_C, the module that uses the axial distance to find the minimum and maximum distance of the axis coordinate point in the output from CENTERLINE.

The main program then calls the VEHICLE, LOG_VEH, ORDER_VEH, DIRECTION, and SPEED modules to read the vehicle data and process them to determine the individual vehicle speeds and average speeds of cars and trucks. The commands to call these modules and associated comment are in lines 68 through 100. These are 12 command lines to call modules in these lines are only 12. These 12 lines call 7 modules, 5 of which are called twice, once for cars and once for trucks. Some of the call command lines take more than one line of program list lines due to the large number of variables being passed to and from these modules and due to the length of the variable names. (Most of the command lines that call modules require more than one program list line and there are comment lines that explain the program within the command lines.)

Next the main program calls CNT_TYPE, the module that asks the user to choose between computing traffic measures during a time interval or computing the time interval for the given highway limits. It does this by asking the user to respond with 1 to compute traffic measures during a time interval and with 2 to compute the time interval for the given highway limits. CNT_TYPE also accepts the user's response. Depending on the user's choice, the main program calls different sets of modules. The flowchart in Figure A-6 depicts the options. Line 104 in Appendix A1 is where the call is made to the module that gives the user the choice and reads the user's response. If the user chooses "1", the main module calls the modules to compute the traffic measures for the given count location, and lines 106 through line 124 are processed. If the user chooses "2", it calls the modules that compute the time interval for given highway limits, and lines 126 through line 187 are processed.

Highway Axis Module

The CENTERLINE module, which process the highway axis data to compute linear distances along the axis, is listed in lines 1 through 74 of Appendix A2. This module reads the coordinates of the points that define the highway axis contained in Highway Axis Data Input file and computes the distances of these points from the same reference datum as the first point in the file. The x and y coordinates of the points are saved in arrays XC and YC. The distances at these axis points are saved in an array, LOC_CL. The coordinate values and the distance for a given point are saved at the same reference location in their respective arrays.

The XC, YC, and LOG_CL arrays are sized at the beginning of the main module and the highway axis module. The statement to declare the sizes of the arrays is found in line 6 of the main module (Appendix A1) and in line 5 of the highway axis module (Appendix A2). Presently these arrays are sized to 800 spaces. If there are more than 800 points that define the highway axis, the statements to set the sizes of these arrays should be modified in these two arrays. (The FORTRAN compiler used to compile this software does not allow for dynamic allocation of memory and has problems with global variables. Therefore, we allocate a memory size for the arrays at the beginning in the main module. For the same reason we allocate the memory size at the beginning of each module for the arrays that are being used in that module.)

After reading the data and assigning distances to the coordinate points, the axes are extended at the edges and an extra point is added to each end of the highway axis. The axes are extended so that vehicles that lie around the beginning or end of the axis can be projected to the axis. This extension becomes important when the highway axis is at an angle with reference to the coordinate axis of the images. (This case is explained in more detail in the LOG_VEH module section.) The need to do this will become clear when we explain the method of assigning distances to vehicles with reference to the highway axis. To allow for these "extensions", the first place in each array is saved for the extension of the beginning of the highway axis. The extension at the end of the axis is saved in the place following the last point of the axis.

CENTERLINE first asks the user for the name of the file that contains the centerline data in line 13 and accepts the user's response in line 14 (see Appendix A2). After reading the name of the file, the CENTERLINE module calls the command to open the file (line 16 of Appendix A2). If the file is opened with no problem, lines 26 through 68 are processed. Otherwise, a failure message is printed at line 70, and the entire program is terminated. When the file containing the axis data is opened, the value in the first line of the data file is read (line 26) and saved in the second space in the array of centerline distances. As explained above, this number represents the distance from some exterior datum to the first axis point, the coordinates of which are listed in the second line of the axis data file. As mentioned above, the first space in the LOG_CL array is kept vacant to save the distance at the extended point of the axis.

Next, CENTERLINE reads the coordinates of the highway axis points in a loop (lines 29 through line 38 of Appendix A2). After reading the first line of the data file the loop starts. The x and y coordinates of each point are read and saved into arrays XC and YC sequentially through this loop. While reading the data the module checks for invalid data. Any data other than numerical values are considered invalid. Alphanumeric characters or any other symbol characters in the data are considered invalid data. Similarly numerical data with more than one decimal point, for example 2.2.0 or 2.2.0.0 are considered invalid input. If any invalid data are read the program is terminated.

In addition to reading the data and checking for validity, the module checks for the end of file within the loop and counts the number of axis points. The number of axis points is used to define the size of the axis arrays to be used to save the data and to read data from. A counter is used to count the number of axis points and this counter increments by 1 every time a new coordinate set is read. When the end of file is encountered the counter stops incrementing and the loop is terminated. These checks are performed through decision statements listed in lines 31 through 38.

When the loop is terminated two extra data records are added to the array. The first is added at the first location, and the second is added at the location following the last record in the array. These records are for the extension of the axis. The beginning of the axis is extended by creating a point located at a distance from the first point of the axis data that is equal to three times the linear distance between the first two points of the input data. The end is also extended in a similar manner, by creating a point located at a distance from the last point of the axis that is equal to three times the linear distance between the last two points of the axis. The beginning of the axis is extended by adding x and y coordinates to the first space in the arrays XC and YC. This is done in lines 43 and 44 of Appendix A2. The last point is extended by adding x and y coordinates to the spaces following those where the last point of axis had been saved. This is done in lines 47 and 48.

The distance read from the first line in the axis data input file was assigned to the second space in array LOG_CL because the coordinates of the point with this distance (i.e., the second line in the axis coordinate data file) are saved in the second spaces of arrays XC and YC. Given the coordinates of this point and those representing the extension of the axis explained above, the Euclidean distance of the extended chord is computed. This distance is subtracted from the distance of the first axis point to yield the distance at the extended first point of axis. The distance is saved in the LOG_CL array in the first space. The software then processes a loop (lines 61-68 in Appendix A2), beginning with the third point, that computes the distances of each point and saves them at the appropriate locations in the distance array, LOG_CL. The distances are determined by computing the Euclidean distance between each point and the previous point and adding this incremental distance to the cumulative distance of the previous point. The logic of this module is illustrated in the flowchart shown in Figure A-7.

Within the same loop (lines 61-68) the module checks for the largest distance in the x or y direction between two consecutive points. This distance is used later in the module that computes the distance of vehicles along the road axis. The largest distance is assigned to a variable called DINC. The module initializes DINC to zero (line 10). Whenever, the loop increments to compute the distance at a point on the axis, the linear distance between the present point and the previous axis point is checked to determine if it is larger than DINC (lines 65 and 66). If the distance is larger than DINC, this distance value is assigned to DINC. When the loop is terminated, the value of DINC is the largest difference in either x or y direction between the coordinates of consecutive points. This value is saved and passed to the main program.

When completed, CENTERLINE returns the control to the main program. It also returns the values of the axis coordinates, the distances along the axis, and DINC to the main module of the software. After completing the CENTERLINE module, the main program calls the MINMAX_C module that determines the minimum and maximum values of the array LOG_CL. These values are needed in later modules. They are saved in variables CMIN and CMAX and passed to the main program.

Vehicle Modules
There are six modules that read vehicle data and process them to obtain vehicle speeds and then the average speed of each class of vehicles. We call these modules the vehicle modules. The first vehicle module is called only once. The other five are each called twice, once for processing car data and again for processing truck data. The first module, called VEHICLES, reads the car and truck data and saves them in arrays. The other five modules use these arrays to determine distance and speeds of cars and trucks. The flowchart shown in Figure A-8 illustrates the order in which these modules are called. The first of these five modules is LOG_VEH. This module uses the vehicle data arrays and the centerline data to compute locations of vehicles, represented as distances, along the centerline. This module is called once for each class of vehicles, cars and trucks in this research. Module ORDER_VEH is called next for each class of vehicles. This module sorts the vehicle data by their ID numbers and returns the vehicle data in the sorted format. After sorting the vehicle data the DIRECTION module is called. This module returns a value of +1 for the variable DIRECT if the distances of the vehicles increase as they travel downstream, otherwise it returns a -1 for the value of the variable DIRECT; that is, a +1 if the distances are measured in the direction of traffic flow and -1 if the distances are measured opposite to the direction of flow. This is important in computing the speeds of vehicles to ensure that the speed values are all positive. It is also important when estimating the times when vehicles pass the count location. We explain this in more detail when we explain the modules that estimate the time when vehicles pass the count location. Once the direction of the increase in the vehicle distances is determined, the SPEED module is called to compute the speeds of the vehicles. Again, SPEED is called once for each class of vehicles. After the speeds of individual vehicles have been computed, module AVG_SP is called. This module computes the average of all the speeds of the vehicles. It computes the average speeds of each class of vehicles separately and is called once for each class. The commands to call the vehicle modules are listed in lines 68 through 100 of Appendix A1. Next, we describe these modules in more detail.

VEHICLES Module. This module reads the data in the format explained in the VEHICLE RECORD DATA section. Every vehicle has a record for every time when it was imaged. The record contains the vehicle identification number, x and y coordinates of location of the vehicle, and the time when the vehicle was at that location.

The code for this module is listed in lines 76 through 132 of Appendix A2. This module first asks the user for the name of the file that contains the vehicle data (line 88). After reading the name of the file input by the user (line 89), the module calls the command to open the file (line 91). If the file is opened without problem, lines 94 through 129 are processed. Otherwise, a failure message is processed and printed (line 132), and the program is terminated.

When the file is successfully opened the counters for cars and trucks are set to initial values of 1(lines 95 and 96), and a loop to read the data is executed. (The counters are defined by variables CRS and TKS for cars and trucks, respectively.) One large loop (lines 98 through 124) is executed once for each class of vehicles. This loop starts by reading the class of the vehicles and, depending on the value of the class, one of two smaller internal loops is executed. If the class is 1, the loop that reads the car data is executed (lines 102 through 107), and if the class is 2, the loop that reads the truck data is executed (lines 110 through 115).

The loop to read car data starts by reading the first car identification number, the x and y coordinates of the car location, and the time when this car was at this location. The time is given in a format consisting of three numbers that represent the hours, minutes, and seconds. The time is then converted to units of hours by calling module T_CONV. The car data is saved in the space defined by the counter for cars, which starts with 1, in the arrays CAR_ID, XCAR, YCAR, and CAR_TIME_ID. The values saved in these arrays are the car identification number, x coordinates of the car location, y coordinates of the car location, and the time in the units of hours. If the car identification number is not –1, the counter for the number of cars is incremented by one (line 106) and the loop is repeated. The next time through the loop the data of the next vehicle is read and saved in the arrays at the location defined by the counter. If the car identification number is –1, which indicates the end of car data records (see the Highway Axis Module section), the loop terminates.

The loop to read the truck data is similar to the loop that reads the car data. The truck data is saved in the arrays TRK_ID, XTRK, YTRK, and TRK_TIME_ID at the locations defined by the counter for trucks. The values saved in these arrays are the truck identification number, x coordinates of the truck location, y coordinates of the truck location, and the time in the units of hours when the truck was at that location.

After both the car and truck data are read, the larger loop is terminated and the module passes the data to the main program. This module presently considers only two vehicle classes, cars and trucks. It can be expanded to accommodate more classes of vehicles. More loops can simply be added to read data for more classes. The new loops would have to be added within the larger loop that contains the smaller read loops.

LOG_VEH Module. The module LOG_VEH computes the linear distances (i.e., distances measured along the road axis) of the vehicles with respect to the externally defined datum. The input data for this module are the arrays that contain the highway axis and vehicle data and the value of DINC. (Recall that DINC was defined in module CENTERLINE above and represents the largest distance in the x and y direction between two consecutive points on the axis line.) The LOG_VEH module passes back to the main program the array of linear distances that represent the vehicle locations along the highway axis. To calculate the linear distance of a vehicle, a perpendicular to the centerline is projected from the x and y coordinates of the vehicle to the centerline axis. Then, the distance from the external datum to the point where the perpendicular line intersects the axis is computed and assigned as the vehicle location distance.

A flow chart of this module is presented in Figure A-9. The code for this module is listed in lines 1 through 181 in Appendix A3. The distances of all the vehicles are computed through a loop that repeats once for each vehicle record.

To determine the distance of a vehicle, the road axis points that are within a given proximity of the vehicle location are identified. The module defines a search proximity box with the vehicle location coordinates in the center and a width and height that are equal to 4 times DINC, which was determine in module CENTERLINE. Any chord that is partially within the search box is inspected. Imaginary perpendicular lines to these chords are drawn from vehicle location. The point of intersection between the perpendicular line and the chord or its extension is determined by calling module INTERSECT (line 26 of Appendix A3). If the point of intersection between the chord and the perpendicular is on the chord, this is defined as the point to reference the vehicle by. If the point of intersectin is on the extension of the chord, the chord is disregarded, and the next chord is checked.

To illustrate, consider the schematic of a highway axis and a car represented in Figure A-10. In this figure highway axis is represented by points C1, C2, C3, and C4 by the chords (C1,C2), (C2,C3), and (C3,C4), where C1, C2, C3, and C4 are the points whose coordinates are saved in arrays XC and YC that represent the highway axis. The car location is represented by the center of the rectangle labeled CAR1. The perpendicular drawn from the car location to the chords (C1,C2), (C2,C3), and (C3,C4) or their extension are points X1, X2, and X3, respectively. Points X2 is on chord (C2,C3), while X1 and X3 are on the extension of the chords (C1,C2) and (C3,C4), respectively. Therefore, we consider point X2 to represent the location of the vehicle. We determine the distance of CAR1 location as being the distance at C2 added to the Euclidean distance between point C2 and X2.

This process is done through a loop that goes through many checks. Lines 31 through 174 are the list of the different check code lines for the intersection point of the two lines. When the intersection is determined on axis chord, module D_LOG is called to compute the distance along the intersection point on the axis. This is done by adding the Euclidean distance from the intersection to the chord edge point to the distance at the end of the chord. This distance is then assigned to the vehicle as its location distance.

Lines 15 through 178 are the commands that process the loop to find the distance location of one vehicle. The large loop determined by lines 12 through 179 is processed once for each vehicle. When all the vehicle distances are computed, the module passes the new vehicle records to the main program. The new vehicle records contain the vehicle identification number, the vehicle distance along the highway axis, and the time when the vehicle was at this location. Figure A-11 presents car record data, for the vehicles in Figure A-2, in the format passed from this module to the main program.

Module INTERSECT takes the coordinates of the end points of the two lines, the highway axis chord line and the perpendicular line, as input and returns the coordinates of the intersection point. This module listed in lines 183 through lines 197 uses basic trigonometry to find the intersection of two lines. It takes line 1, which represents the chord on the highway axis, and line 2, which represents the perpendicular to the chord from the vehicle location, and finds their intersection. Line 1 is defined by coordinates (x1,y1) and (x2, y2) and line 2 is defined by coordinates (x3,y3) and (x4,y4). Point of intersection is defined by point (x5,y5) and the equation to compute these coordinates are listed in lines 194 and 195.

In determining the vehicle location distance with reference to the road axis for the vehicles that lie at the beginning or end of the axis, the perpendicular may intersect at a point on the first chord outside the axis limits. When the axis of the highway is at an angle with reference to the coordinates of the first image, locations of some vehicle could be out of the range of the axis. This case is represented in the schematic of Figure 12. The schematic represents a case of a first image in a series of images. The axis of the highway in this image is at a sharp angel with the respect to the image X axis of the image. Truck-1 is out of the ranges of the highways axis. When a perpendicular is dropped from the location of Truck-1 to the axis, the intersection of the axis and the perpendicular lies outside the ranges of the image limits and thus the range of the axis.

This case is treated in our work by extending the axis beyond the starting point at the limit of the image. This extension should be long enough to ensure that the intersection of the axis and the perpendicular on the axis of the highway lie on this extension.

Recall, we explained in the Highway Axis Module section that the highway axis are extended at the ends to consider the vehicles that may lie at the beginning and end of the highway. This was the reason for extending the axis at the beginning and the end in the CENTERLINE module.

ORDER_VEH Module. Module ORDER_VEH sorts the vehicle data in ascending order of vehicle identification number. The new sorted vehicle data and identification numbers are saved in new arrays. Vehicle data are ordered such that vehicles with similar identification numbers are in consecutive locations. Figure A-13 presents the vehicle records of Figure A-11 in the new format.

The general process of this module is presented in the flowchart shown in Figure A-14. The code for this module is listed in Appendix A3 in lines 229 through 297. As seen in the flowchart, we determine a vehicle to be the present vehicle under consideration. We call the vehicle that is being processed the present vehicle and use the variable LATESTVEH to indicate the ID of this vehicle. We start the loop by defining the present vehicle to be the vehicle with the smallest identification number of all the vehicles in the class (line 254 in Appendix A3). The smallest identification number is defined by calling module MINMAX with the array that contains vehicle identifications (line 249 of Appendix A3). This array returns the smallest and largest vehicle identification numbers.

The vehicle records are sorted through two nested loops. The outer loop changes the present vehicle ID every time the loop is incremented. The inner loop checks the entire set of vehicle records to find all the vehicles with the same identification number. Each vehicles with identification numbers identical to the present vehicle identification number is saved in a new array N_VEH_ID in the order that it is found in VEH_ID each in the next available cell. At the same time these vehicles are marked for deletion in the old array VEH_ID of identification number. These vehicles are marked for deletion so that this cell will not be checked the next time we go through the array to check a different vehicle identification. When the last vehicle in the array VEH_ID has been checked to find all the vehicles with identical ID as the present ID, the LATEST_VEH variable is incremented (line 266) and the smaller loop is terminated. The larger loop checks for the LATEST_VEH to be less than or equal to the largest vehicle ID. When an ID greater than that of the LATEST_VEH is found there are no more vehicles left to be ordered, and the larger loop is terminated.

As the vehicle identification numbers are saved, their distances and time data are also saved in the same reference location in new arrays N_VEH_LOG, and N_VEH_TIME_ID, respectively. This module process all the vehicle data and passes the new set of arrays that contain the vehicle data sorted by vehicle identification number to the main program. These new vehicle data are used in the next modules.

SPEED Module. This module computes the speed of every vehicle that is listed more than once in the vehicle data. A vehicle is repeated more than once when its identification number is repeated more than once in the list of identification numbers. This would be the case when the vehicle is imaged more than once. Vehicles that do not appear more than once are given a speed of zero. The speed of every vehicle is saved in a new array called VEH_SP in the same reference location as that of the corresponding vehicle as the other arrays. The vehicle location and identification are saved in N_LOG_VEH and N_VEH_ID in a location marked by the vehicle counter. The speed is saved in the array VEH_SP at the location marked by the same counter. The data used in this module are the sorted data that were passed from module ORDER_VEH. The process of this module is presented in the flowchart of Figure A-14. The code for this module is listed in lines 290 through 312 of Appendix A3.

Speeds are computed in a loop that starts at the second location in the vehicle identification array. If the identification of the vehicle in this record is equal to the identification number of the vehicle in the first location, the speed of the vehicle is computed and saved in array VEH_SP. Otherwise, the vehicle is assigned a speed of zero and the module proceeds to process the next vehicle. Only consecutive vehicle identifications have to be checked because the vehicles have been ordered in the previous module such that the consecutive appearances of the same vehicle are in consecutive locations in this list.

Speed is computed by dividing the difference in the location distances by the time difference of these two vehicle locations. Recall that the distances are linear distances, since the vehicles locations were projected to the axis in module LOG_VEH. The calculated speed represents the average speed between these two locations during the time when the vehicles were imaged at these locations. The speeds of the vehicles are saved in the array in the same reference location parallel location to the second appearance of the vehicle. The speed in the location referenced by the same reference location as first appearance of the vehicle is given a zero in the speed array. The speed of the vehicle is computed in line 303 in Appendix A3. In the present version, the distance of vehicles is assumed to be meters and the time in hours; therefore, speed is divided by 1000 (line 303) to convert the speed to units of kilometers per hour (KPH).

This module passes the array of speeds of vehicles to the main program. These speeds are used in later modules.

AVG_SPD Module. This module computes the average speed of all the vehicles in the array that contains the speed data. This module calculates the average speed as the sum of the speeds divided by the number of non-zero speed values. This gives the average speed of the vehicles in the class for which the data are being processed. This average corresponds to the space mean speed of the vehicles. The code for this module is listed in lines 314 through line 329 of Appendix A3.

This space mean speed is then substituted for the speed of vehicles that have been imaged only once. The speeds of these vehicles had been temporarily set to zero. Recall that the speeds of speeds of cars are generally greater than speeds of trucks; therefore, substituting the average speed of cars for the speeds of a cars would tend to lead to more accurate results than when substituting the average speeds of all the vehicles. Similarly, substituting the average speed of trucks for the speeds of trucks would tend to lead to more accurate results than when substituting the average speeds of all the vehicles. For this reason, we compute the average speed of each vehicle class separately by calling this module to compute the average speeds of cars once and to compute the average speed of trucks once.

Count Type

This is a simple module that asks the user to enter the choice of modules to run. It requires the user to enter a 1 to compute traffic measures at a given location and time interval or to enter a 2 to compute the largest time interval for which parameters can be estimated for the given road location highway limit. The user might not be able to define the count interval from the data, in this case the user can define the highway limits from the image and determine the maximum count interval that this highway limits would allow. This interval then can be used to determine the count intervals, within this interval, that the user wishes to use to get traffic parameters.

This is the module that represents the choice in the general flowchart of the program shown in Figure A-15. This module is listed on lines 1 through 37 of Appendix A4. If the user enters a 1 or a 2 as a response, the module returns the control to the main program and passes the response back too. If the user's response is anything else other than a 1 or a 2, a message is presented to indicate that the response is invalid, and the response is requested from the user again.

According to the user's response, different sets of lines are processed in the main program. When the user's response is 1, lines 107 through 148 of the main program, listed in Appendix A1, are processed. These lines call a series of modules called COMPUTE-1. When the user's response is 2, lines 151 through 188 are processed. These lines call a series of modules called COMPUTE-2.

COMP -1 Modules

Compute modules are modules that read the highway data file and compute traffic m at the given location during the given time interval. The flowchart presented in Figure A-16 shows the general process of this set of modules.

In COMPUTE-1, traffic measures are computed from the estimated times of when the vehicles pass the count location. Since this work is motivated in large part by a desire to compare measures estimated from satellite images to those that would be estimated from an ATR (Automatic Traffic Recorder), we refer to the count location a ATR location. The count location does not have to correspond to a true ATR location; it could be any location on the given highway. This name is used for simplicity to identify the count location.

The times when the vehicles would pass the ATR locations are estimated from the given location and time data in arrays N_VEH_LOG and N_VEH_ _ID. When a vehicle has more than one location and time data, the closest location of the vehicle to the ATR location is used to estimate the time when it would pass the ATR location. The user can either use the average speed of the vehicles of the class or the speed of the individual vehicle at the location where it resides to estimate the time it would take to travel from the given location to the ATR location. We use these options to compare the measures that we estimate using each speed to check the accuracy of both versus the measures estimated from an ATR.

In COMPUTE-1 series of modules, the first module called is XYATR, which reads the highway data from a data file and passes the data back to the main program. Then, the user is given a choice of which speed to use to project the vehicles to the ATR location. If the user chooses to use the average speed of vehicles, then the BRING_TO_ATR_A_SP module is called twice, once with truck data and once with car data. Otherwise, if the user chooses to use individual speeds of vehicles, BRING_TO_ATR module is called twice, again once for truck data and once from car data. The BRING_TO_ATR and BRING_TO_ATR_A_SP modules pass the estimated time when the vehicles pass the ATR location to the main program. When these modules are completed, the COMP_PAR module, which computes the parameters and prints them, is called.

In the following sections we present details of the modules used in COMPUTE-1 in the order that these modules are called.

Count Locations Module. This module is called XYATR and it is listed in lines 1 through 48 of Appendix A4. XYATR first asks the user for the name of the file that contains the count location data (line 15) and accepts the user's response (lines 17). After reading the filename, the module calls the command to open the file (line 21). If the file is opened successfully, lines 21 through 42 are processed. Otherwise, a failure message is printed at line 44, and the module and the entire program are terminated.

When the file is opened, the module reads the data. The number of lanes and the passenger car equivalent of trucks are read and assigned to variables NL and Et, respectively, in lines 21 and 22. Line 23 reads the ATR location x and y coordinates, start of count interval, and end of count interval. Each of the times is read in three numbers that represent hours, minutes, and seconds. Module T_CONV is called to convert each of the times to one number in hour units. This module, called twice (lines 29-30), converts each of the times - count start time, and count end time - to hour units. These times are returned as values of the variables T1 and T2.

Module LOG_LOCATION, called in line 34, computes the count location distance along the highway axis and assigns it to DIS_ATR. After determining the count location distance, this XYATR module terminates and passes all the data to the main program.

BRING_TO_ATR_A_SP Module. The BRING_TO_ATR_A_SP module estimates the time when each vehicle passes the ATR location using the average speed of the vehicles of the class of the vehicle being estimated. When a vehicle has only one location record, this location is used to estimate the time when it passes the ATR location. When a vehicle has more than one location record, the location closest to the ATR is used to estimate the time when the vehicle passed the ATR location. The time when the vehicle was at the location of the ATR is computed by estimating the time that the vehicle would take to travel from the defined location to the ATR location and adding this time to the time when the vehicle was imaged at the location of record closest to the

ATR. The speed of the vehicle while traveling to the ATR location is the average speed of the vehicles of the class of vehicles that are being processed. Recall that when the user chooses to use the average speed of vehicles, module BRING_TO_ATR_A_SP is called. Figure A-17 presents a flowchart of the process of this module. The code for this module is listed in lines 49 through 200 in Appendix A4. (The data that are used in this module are the data that are sorted in the ORDER_VEH module. Thus, the location and time records of a vehicle are listed in consecutive order.) Line 93 is the start of a large loop that repeats with every vehicle record. Each time through this large loop, a small loop listed in lines 96 through 102 is processed. This smaller loop checks whether the vehicle has more than one record. The first and last records of the same vehicle are determined in this small loop.

If the vehicle has only one record the time when it would have passed, the ATR location is computed in the equation listed in lines 106 and 107. In these lines, atr_t_veh is the variable representing the estimated time when the vehicle pass the ATR location, n_veh_time_id is the variable representing the time when the vehicle was imaged, log_atr is the variable representing the location of the ATR, n_log_atr is the variable representing the location of the vehicle, and spd is the variable representing the average speed of vehicles.

If the vehicle has more than one record, lines 105 through 151 are processed. In these lines first the location of the ATR is checked (lines 114 through 133) to determine whether it is located between any consecutive locations of the vehicle. If this is the case, then the time when this vehicle passed the ATR is estimated using the first one of these two locations for this vehicle. This is done in the loop that is listed in lines 111 through 122. If the location of the ATR is not between 2 consecutive locations of the vehicle, then the location record closest to the ATR location is determined and used to compute the time when the vehicle would have passed the ATR. This is done in lines 123 through line 150.

The new times when the vehicles are estimated to pass the ATR location are saved in a new array called ATR_T_VEH. The minimum and maximum values in this array are determined in line 158 and 159 and saved in variables TMIN and TMAX, respectively. The identification numbers of these vehicles are saved in array ATR_V_ID in parallel locations to their times in the array ATR_T_VEH. When the data of the vehicle with the same identification have been processed, the loop finishes one cycle at line 163 and increments to run the for a vehicle with new identification number. If the last vehicle has been processed, this loop terminates and line 164 is processed. The check for more vehicle data is performed at line 160.

After the times that the vehicles are estimated to pass the ATR locations have been determined, the number of vehicles estimated to pass the ATR location during time interval $[t^1,t^2]$ is determined. This is done in a loop that starts at line 173 and runs through line 179. The vehicle identification numbers and speeds for the vehicles that are in the time interval are saved in the new arrays ID_IN_T and SP_IN_T, respectively. Then the vehicles that have speeds are counted and the average of these speeds is computed. This is done in a loop listed in lines 185 through line 194.

After computing the average speeds of vehicles in the count interval, this module terminates and passes the data to the main program.

BRING_TO_ATR Module. The BRING_TO_ATR module estimates the time when each vehicle passes the ATR location using the individual speed of the vehicle. Recall that when the user chooses to use the individual speed of vehicles, module BRING_TO_ATR is called. This module uses the individual speeds to project the vehicles to the ATR location.

This module works in the same manner as the previous module, BRING_TO_ART_A_SP, except that the speed used to bring the vehicle to the ATR location is the average speed of the individual vehicle. If the vehicle has only one location record and no speed was estimated for this vehicle, the average speed of the vehicles of the class is used in the equation to estimate the time at the ATR. If the vehicle has only one speed record, this speed is used to estimate the time at the ATR location. When a vehicle has more than one speed record, the speed of the vehicle at the location closest to the ATR, as explained in module BRING_TO_ATR_A_SO, is used to estimate the time. This module is listed in lines 202 through 363 of Appendix A4.

COMPUT_PAR Module. The COMPUTE_PAR module computes the traffic parameters at the given ATR location during the time interval given. Module BRING_TO_ATR or BRING_TO_ATR_A_SP computed the number of cars and the number of trucks that are estimated to pass the ATR location in the given time interval $[t^1,t^2]$. The average speeds of all the vehicles that pass this location in this time interval was also computed. Module COMP_PAR takes this speed and the number of cars and trucks that are estimated to have passed the ATR location during time interval $[t^1,t^2]$ and the highway count data described in the Highway Count Data section and computes traffic parameters. The parameters computed in this module are the volume of cars in time interval $[t^1,t^2]$, the number of trucks, total number of vehicles, percent of trucks, flow in passenger car equivalent (PC), the space mean speed, and the density in vehicles and in PC. This module then lists the output to the screen.

The code for this module is listed in lines 1 through 41 in Appendix A5. Traffic parameters are computed in lines 15 through 23 and printed out in lines 25 through 39. Figure A-18 shows an example of an output printed out by this module.

COMPUTE-2 Modules

Compute modules are modules that read the highway data file and compute the largest count time interval for the given data. In COMPUTE-2, the time interval is determined.

As in BRING_TO_ATR and BRING_TO_ATR_A_SP, explained in the COMPUTE-1 Modules section, the times when the vehicles pass the specified location, ATR location, are estimated. From these times the earliest time and the latest time when a vehicle passes the ATR are determined. These earliest and latest times determine the allowable time interval for the count. Module X1X2 is called to read the count location time and highway limits data explained in the Highway Location and Limit Data section. Modules BRING_TO_ATR_X1X2_AS and BRING_TO_ATR_X1X2 are called to estimate the time when the vehicles pass the ATR location using the average speed of vehicles and the individual speeds of vehicles, respectively. Module CHECK_T1T2_X1X2 is called to determine the maximum allowable time interval for the count.

The first module called is X1X2. This module reads the highway limit data and passes the data back to the main program. (Highway limits data are the coordinates of the first and last location on the highway segment under consideration.) Then the user is given a choice of which speed to use to project the vehicles to the ATR location. If the user chooses the average speed of vehicles, then BRING_TO_ATR_X1X2_AS module is called once with truck data and once with car data. Otherwise, the individual speeds of vehicles are used to project these vehicles to the ATR location. In this case BRING_TO_ATR_X1X2 module is called. Both modules pass the estimated time when the vehicles pass the ATR location to the main program. Then CHECK_T1T2_X1X2 module, which prints out the time interval is called.

X1X2 Module. The X1X2 module is listed in lines 1 through 57 of Appendix A6. Module X1X2 starts by prompting the user for the name of the file that contains the count location data and waits for the user to enter the filename. The commands for this prompt and response are listed in lines 21 and 22 of Appendix A6. After reading the filename in line 23, the module calls the command to open the file. If the file is opened successfully lines 27 through 51 are processed. Otherwise, a failure message is printed at line 53, and the module and the entire program are terminated.

When the file is opened, the module reads the data. The loop listed in lines 29 through line 38 reads the x and y coordinate data for the count location, the beginning limit of the highway, and ending limit of the highway. Module LOG_LOCATION is called next to compute the distances along the highway axis for the location and limits of the highway. After being determined, the location distances are printed and module X1X2 terminates and passes the ATR location and highway limits data to the main program.

BRING_TO_ATR_X1X2_AS Module. The code for the BRING_TO_ATR_X1X2_AS module is listed in lines 59 through 211 of Appendix A6. This module estimates the times when the vehicles pass the ATR location using the average speeds of vehicles. It has the same logic as modules BRING_TO_ATR_A_SP used in COMPUT-1, which was explained in the BRING_TO_ATR_A_SP Module section. It differs in that the estimated time that a vehicle passes the ATR in

BRING_TO_ATR_X1X2_AS is checked against the maximum and minimum times. If the estimated time that a vehicle passes the ATR is larger than the maximum time, this time is set to be the maximum. Similarly, if the estimated time that the vehicle passes the ATR is smaller than the minimum time, this time is set to be the minimum time. The maximum time is determined to be the latest time when the vehicles of the class pass the ATR locations. The minimum is determined to be the earliest time when the vehicles of the class pass the ATR location. These minimum and maximum times are the times to determine the count interval to estimate traffic measures from the given satellite data. The values of the minimum time and the maximum time are passed to the main program when each of the modules terminates. This module is called twice, once for cars and once for tucks.

BRING_TO_ATR_X1X2 Module. The code for the BRING_TO_ATR_X1X2 module is listed in lines 213 through 374 of Appendix A6. This module estimates the times when the vehicles pass the ATR location using the individual speeds of vehicles. It has the same logic as module BRING_TO_ATR used in COMPUT-1, which was explained in the BRING_TO_ATR Module section. As in BRING_TO_ATR_X1X2_AS, this module checks the estimated time that the vehicles pass the ATR location against the maximum and minimum times. If the time that a vehicle passes the ATR is larger than the maximum time, this time is set to be the maximum. Similarly, if the time that the vehicle passes the ATR is smaller than the minimum time, this time is set to be the minimum time. The maximum time is determined to be the latest time when the vehicles of the class pass the ATR locations. The minimum is determined to be the earliest time when the vehicles of the class pass the ATR location. As explained in the previous section, these minimum and maximum times are the times to determine the count interval to estimate traffic measures from the given satellite data. The value of the minimum time and the maximum time is passed to the main program when each of the modules terminates. This module is called twice, once for cars and once for tucks.

CHECK_T1T2_X1X2 Module. The code for the CHECK_T1T2_X1X2 MODULE module is listed in lines 104 through 129 in Appendix A7. This module takes the minimum and maximum times that cars and trucks would have passed the ATR location, which were estimated in modules BRING_TO_ATR_X1X2 or in BRING_TO_ATR_X1X2_AS, and determines the maximum allowable interval for the count. The largest of the minimum car and trucks times is considered the start of the count interval and the smallest of the maximum car and truck times is considered the end of the count interval.

Figure A1. Sample of highway axis data.

```
   2000
1087, 6106
1080, 6077
1071, 6040
1065, 6015
1057, 5982
1048, 5946
1039, 5909
1025, 5852
1018, 5823
1011, 5795
1007, 5775
 997, 5738
 991, 5713
 983, 5677
 976, 5648
 971, 5628
 965, 5603
 960, 5583
 953, 5554
 947, 5530
 937, 5487
 931, 5465
 930, 5459
```

Figure A2. Photographs 94 and 95.
The reference axis of the photographs and the first axis point.

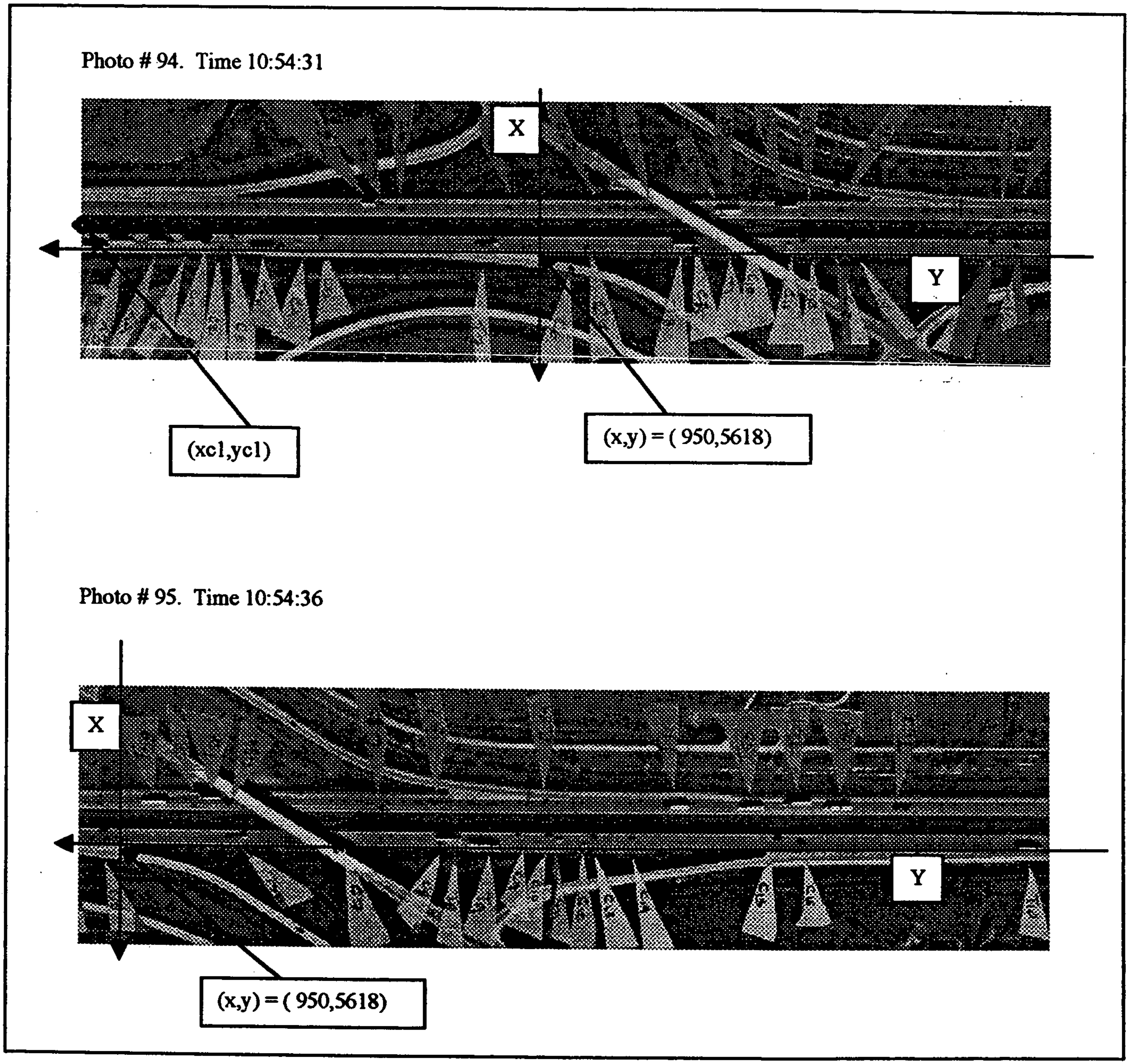


Figure A3. Sample of vehicle record data.

```
1
4,  1018, 5846,  10  54  31
5,  1017, 5828,  10  54  31
7,   959, 5615,  10  54  31
8,   936, 5527,  10  54  31
10,  931, 5481,  10  54  31
11,  921, 5451,  10  54  31
12,  910, 5413,  10  54  31
13,  922, 5439,  10  54  31
14,  914, 5405,  10  54  31
15,  887, 5293,  10  54  31
16,  878, 5272,  10  54  31
7,   922, 5472,  10  54  36
8,   903, 5392,  10  54  36
9,   896, 5331,  10  54  36
10,  890, 5306,  10  54  36
11,  884, 5297,  10  54  36
12,  876, 5278,  10  54  36
13,  880, 5270,  10  54  36
14,  872, 5239,  10  54  36
15,  844, 5121,  10  54  36
16,  837, 5103,  10  54  36
10,  848, 5135,  10  54  41
11,  846, 5144,  10  54  41
14,  832, 5074,  10  54  41
-1, -1, -1, -1, -1, -1
2
3,  1050, 5981,  10  54  31
4,  1042, 5951,  10  54  31
5,  1036, 5921,  10  54  31
6,  1017, 5862,  10  54  31
7,   979, 5687,  10  54  31
8,   939, 5527,  10  54  31
7,   946, 5550,  10  54  36
8,   904, 5379,  10  54  36
9,   892, 5349,  10  54  36
-1, -1, -1, -1, -1, -1
```

Figure A4. Sample of highway count data.

```
3
1.5
903, 5393

10   54    30
10   55    00
```

Figure A5. Sample of highway limits data.

```
1011, 5795

1080, 6077
 947, 5530
```

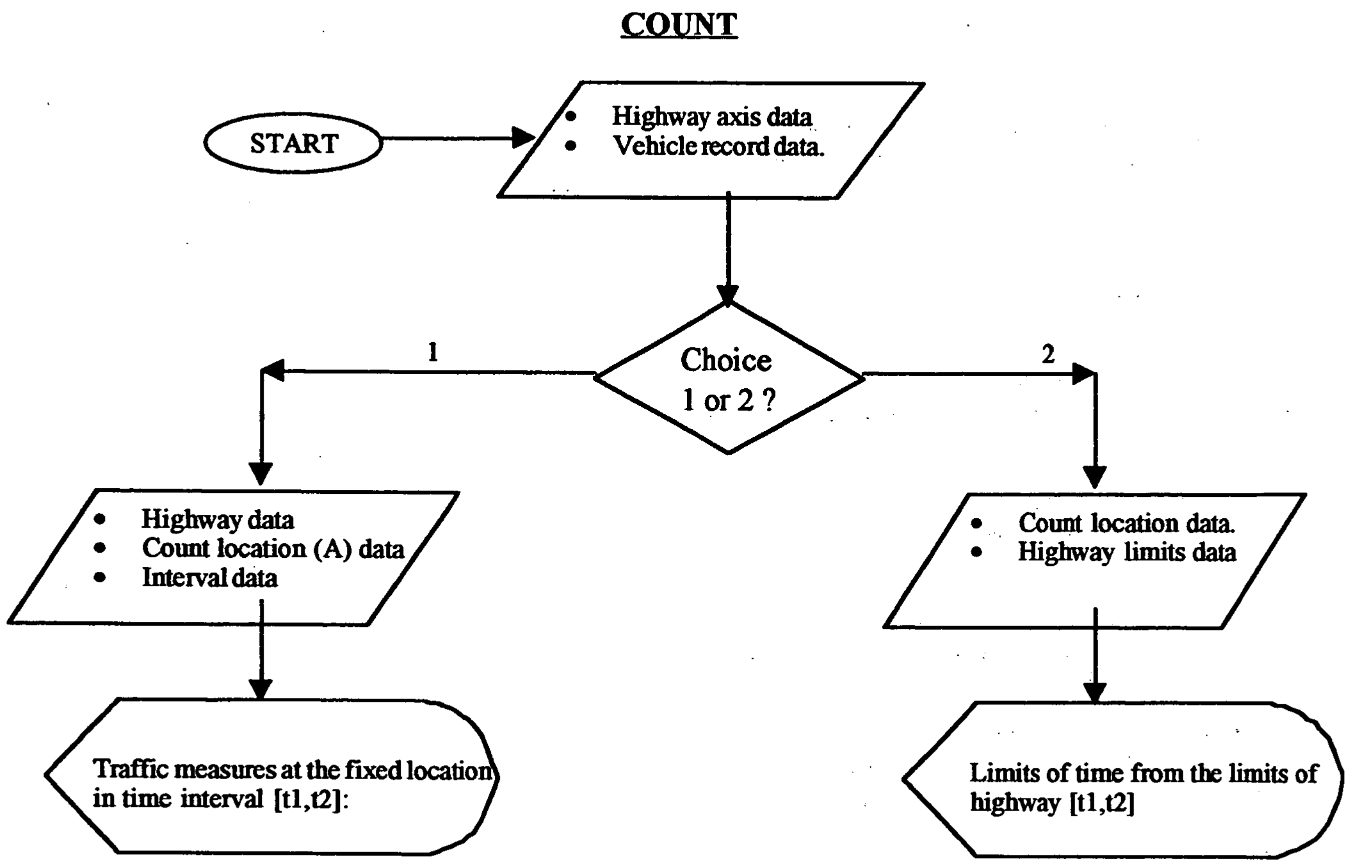


Figure A6. General flowchart of the COUNT software.

Highway axis data include:
- Highway Datum point
- Highway axis coordinates (xc,yc)

Vehicle record data contains:
- Vehicle coordinates (xv,yv)
- Time vehicle was at (xv,yv) coordinates
- Identifier of vehicle at (xv,yv) coordinates
- Class of vehicle at (xv, yv) coordinates.

Highway data include:
- Highway number of lanes Nl
- Truck Terrain factor Et

Count location data include:
- Traffic estimate location (A) coordinates (xa,ya)
- Time point A was imaged

Interval data include:
- Time interval limits [t1,t2]

Count location data include:
- Traffic estimate location (A) coordinates (xa,ya)
- Time point A was imaged

Highway limits data include
- Highway limits coordinates (x1,y1), (x2,y2)
- Time when these limits of the road were imaged.

Traffic measures at the fixed location in time interval [t1,t2]:
- Volume of cars
- Volume of trucks
- Total volume
- Percent of trucks
- Equiv. Of passenger car flow
- Space mean speed
- Equivalent passenger car density

A- AXIS MODULE

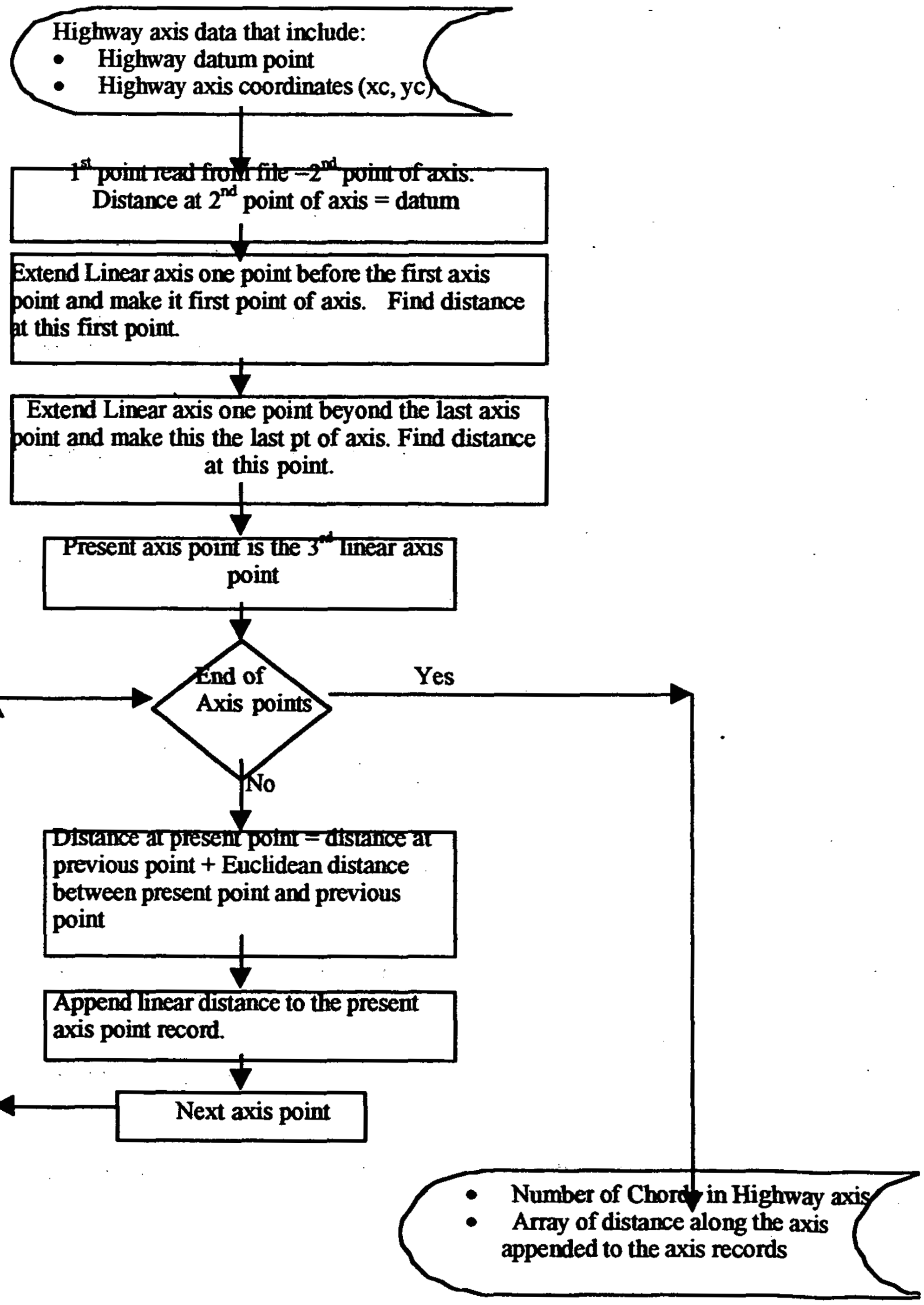


Figure A7. Flowchart of highway axis module.

B- VEHICLE MODULES

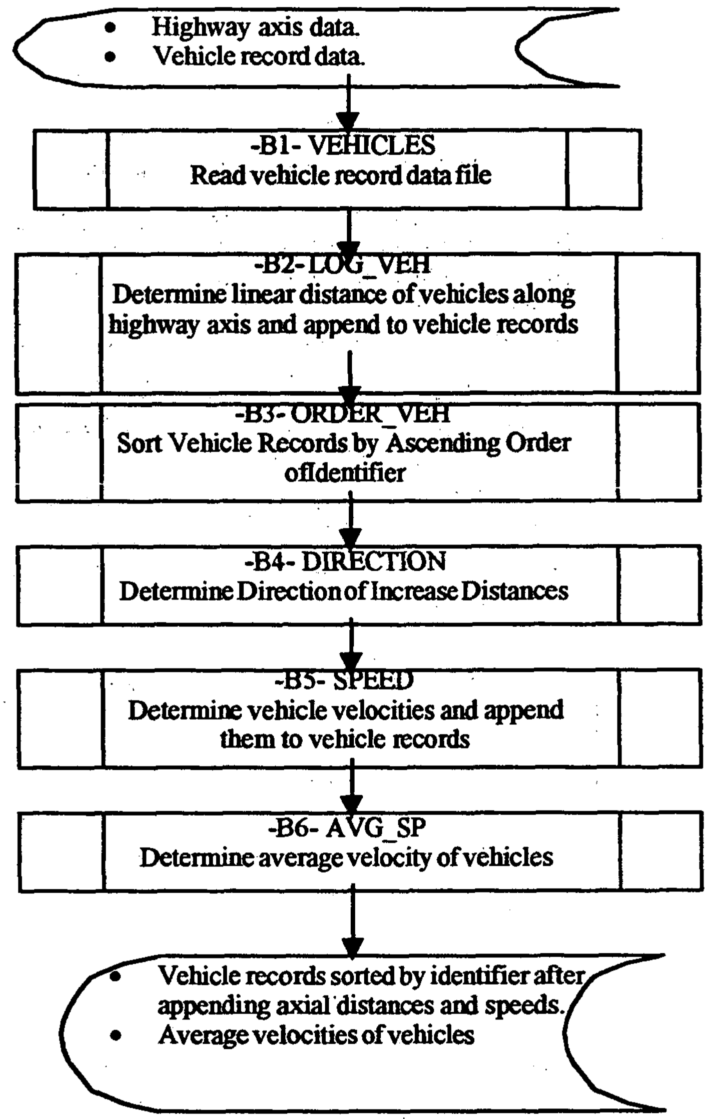


Figure A8. Flowchart that shows the order for calling the vehicle modules.

B- LINEAR DISTANCES OF VEHICLES

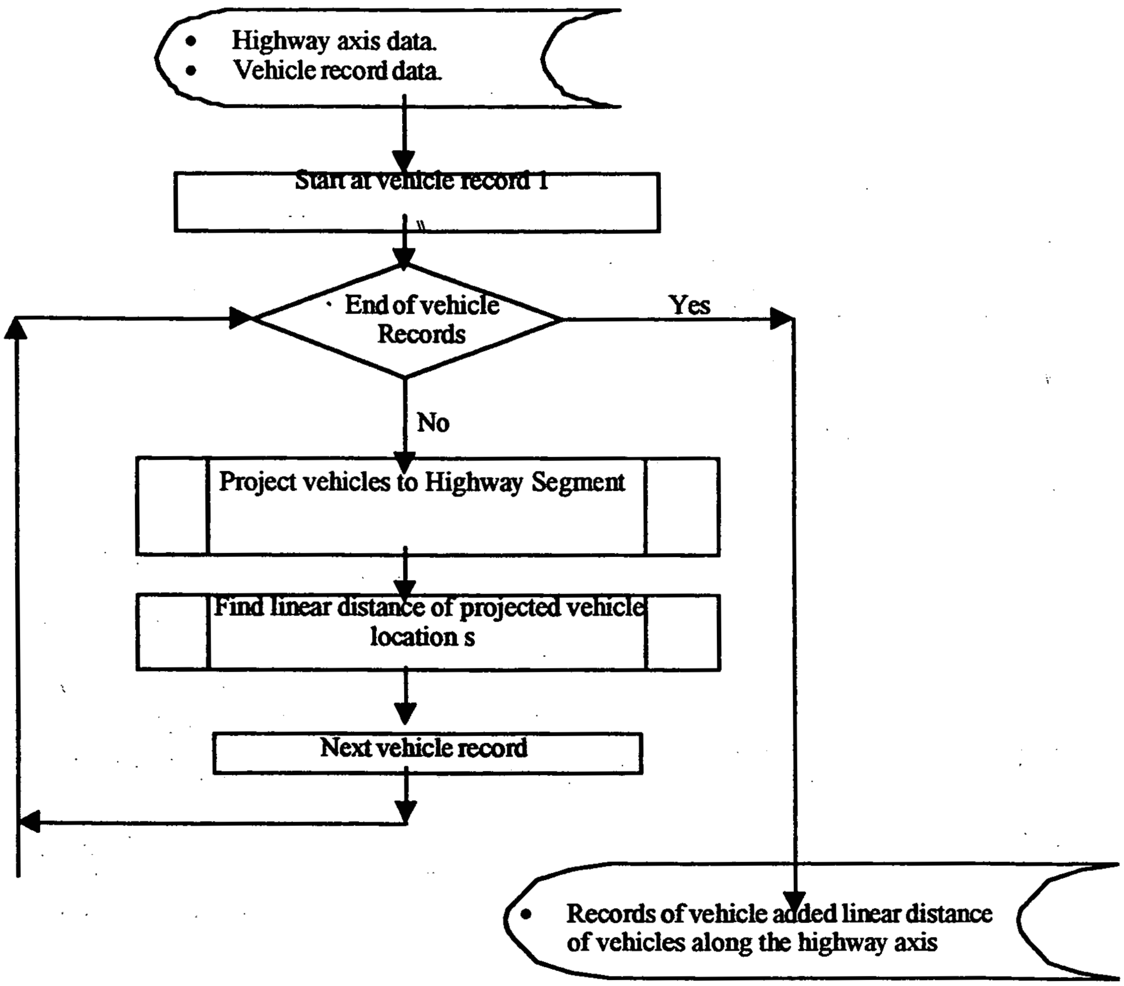


Figure A9. Flowchart of LOG_VHE module.

-B2- SORT VEHICLE RECORDS BY IDENTIFIER

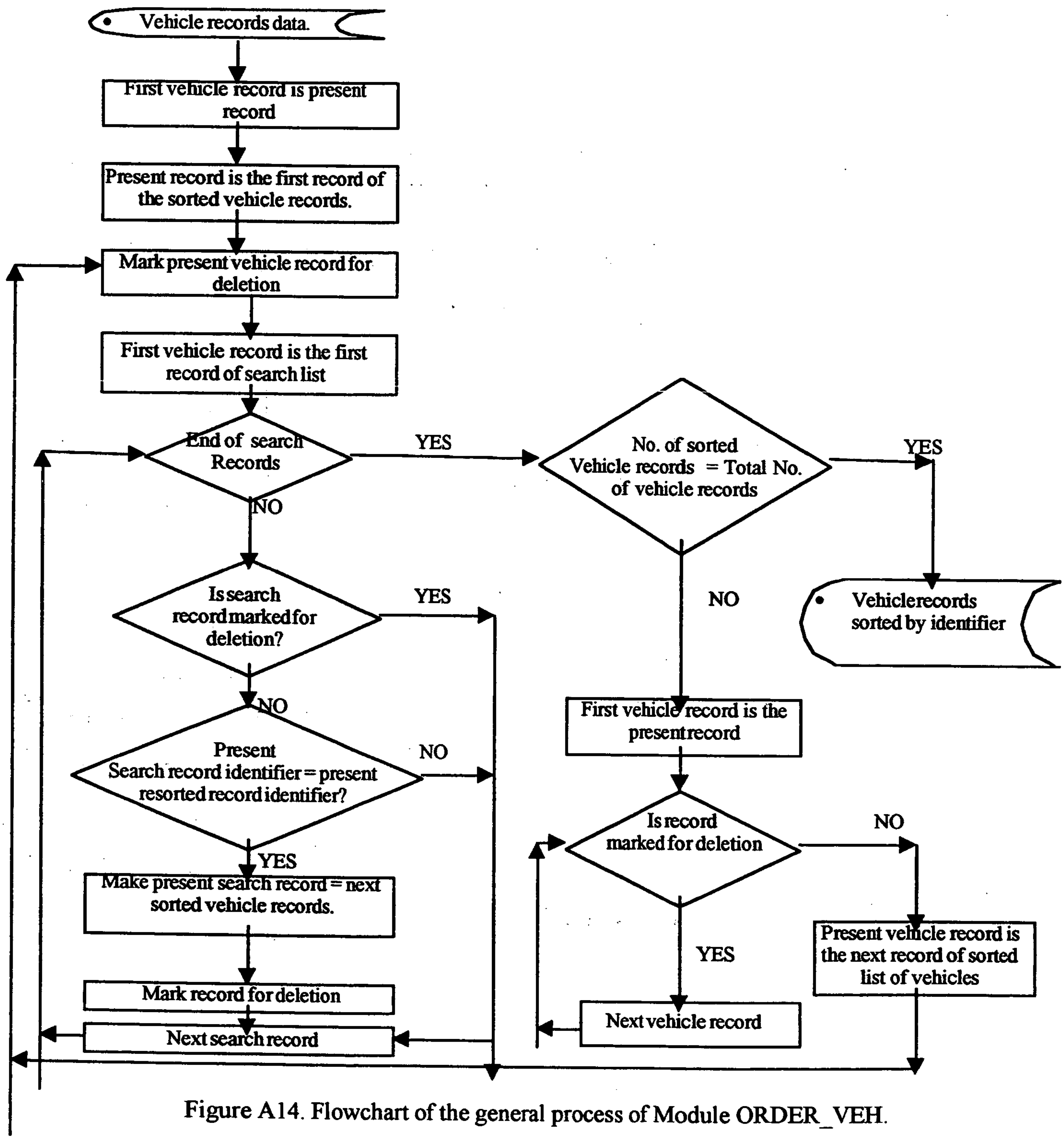


Figure A14. Flowchart of the general process of Module ORDER_VEH.

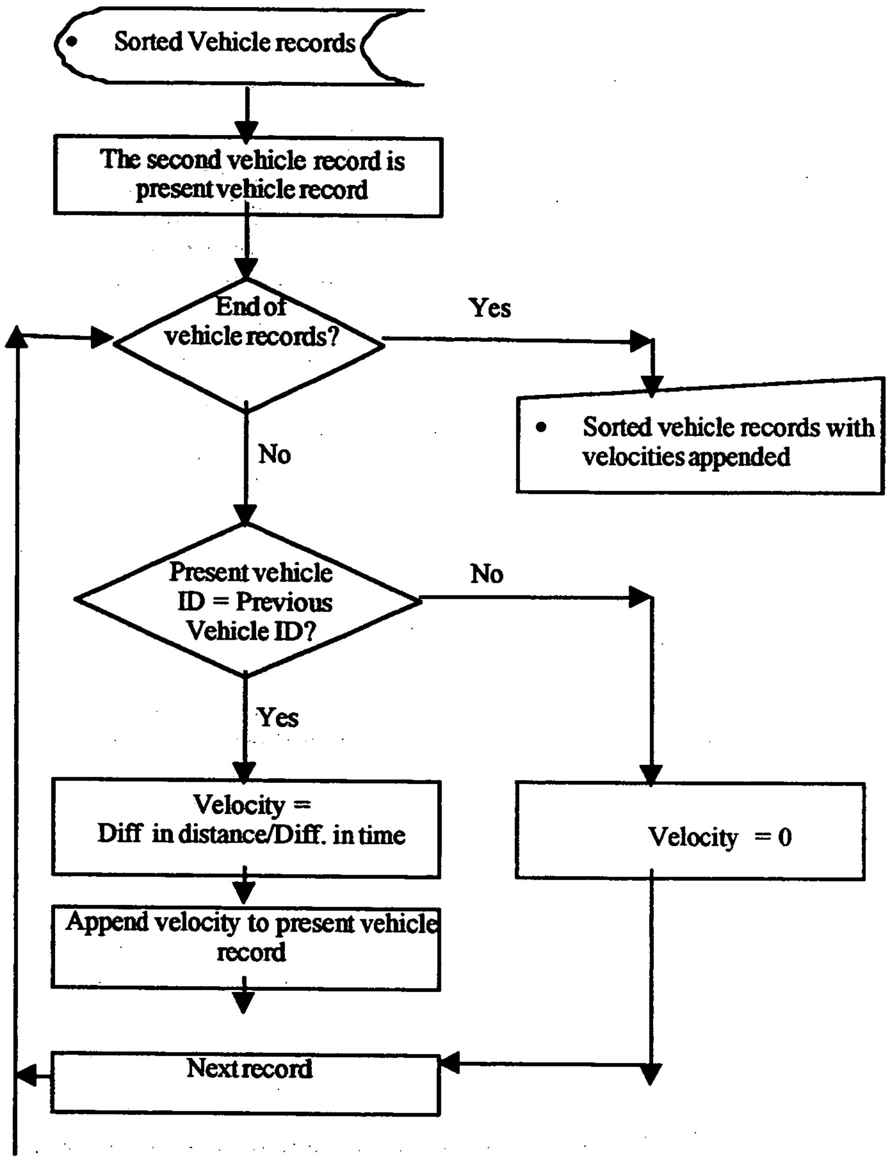


Figure A15. Flowchart of SPEED module.

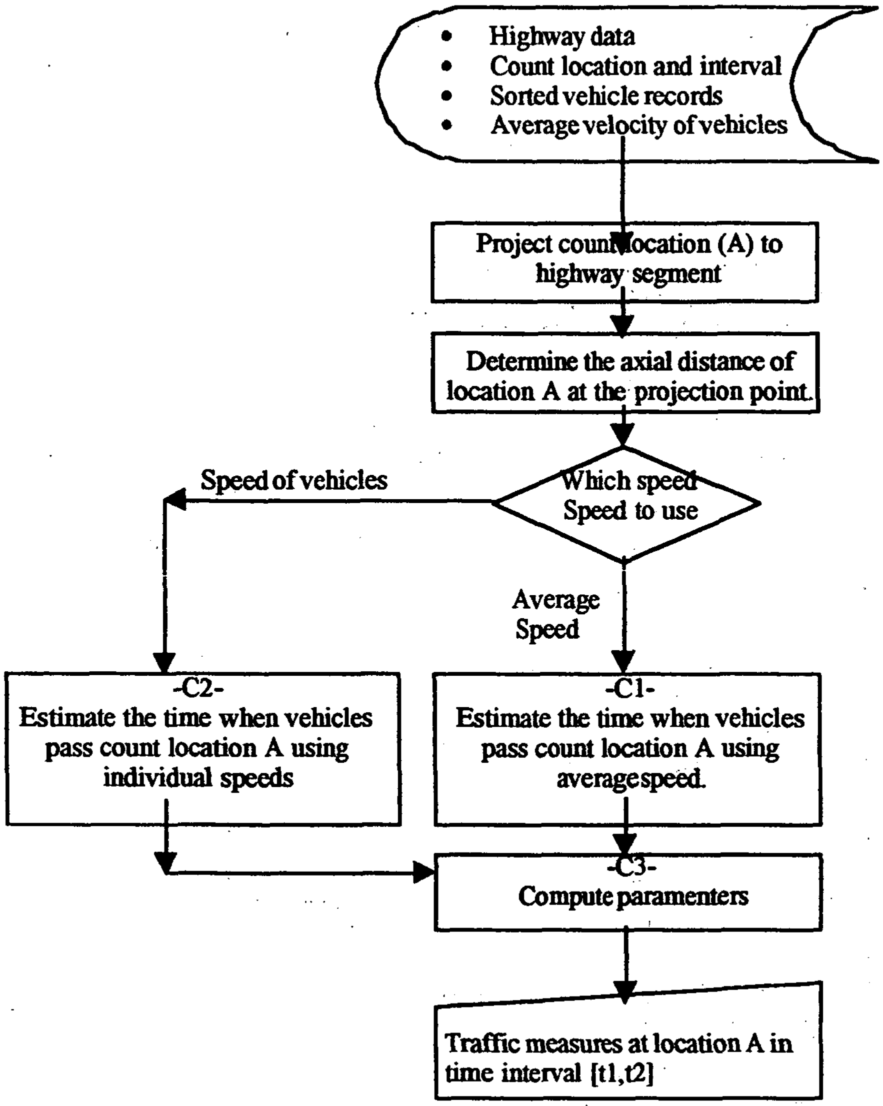


Figure A16. Flowchart that shows the process of calling the set of COMPUTE modules.

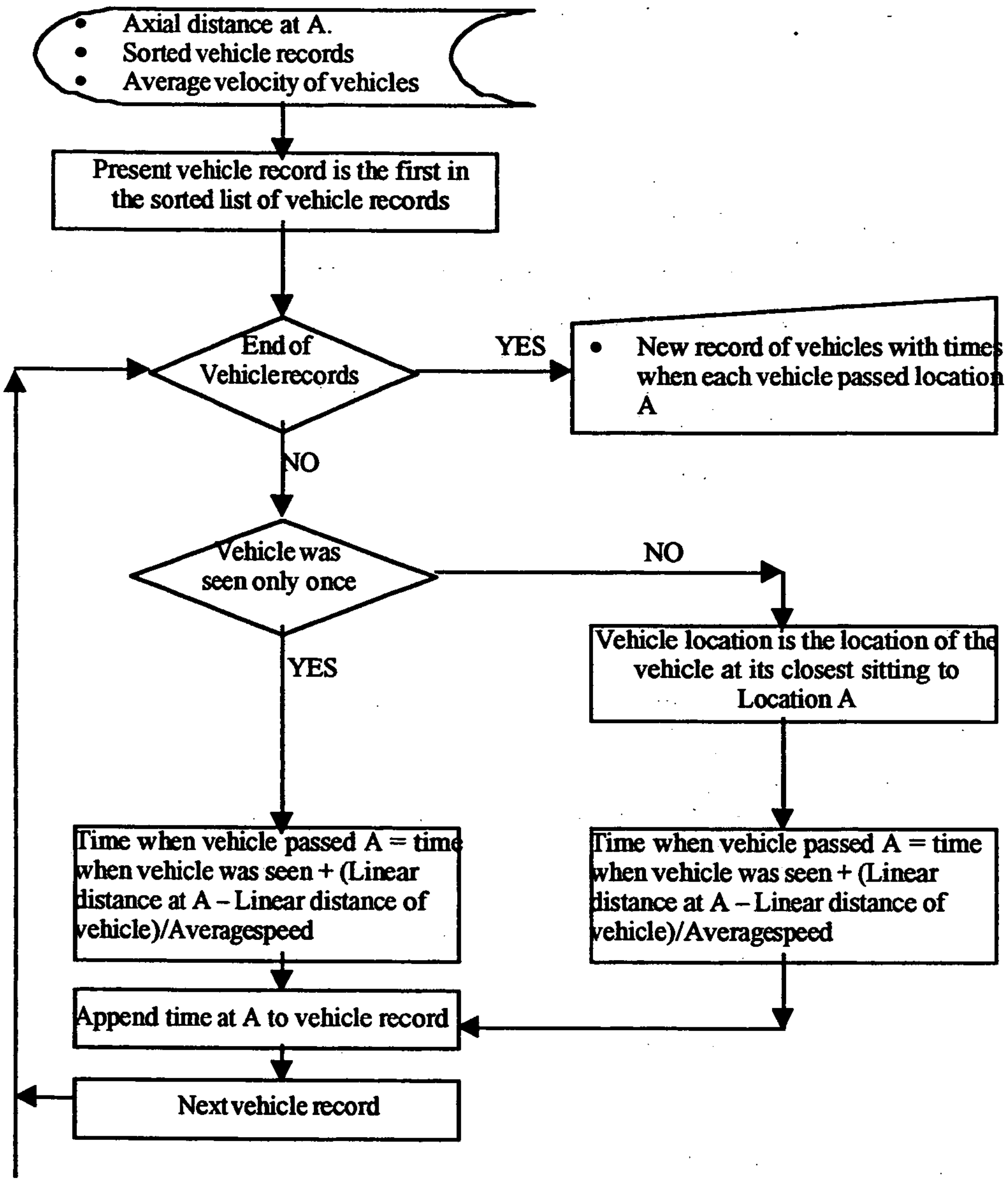


Figure A17. Flowchart of the BRING_TO_ATR_A_SP module.

```
       program count
     ! !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
       implicit none
     ! variables introduce for the first time in centerline subroutine
       integer numcl
       real xc(800),yc(800),log_cl(800), dinc
     ! variables introduce for the first time to find min and max
     ! of CL points
       real cmin,cmax
     ! variables introduce for the first time in Vehicles subroutine
       integer crs,tks,car_id(400),trk_id(400)
       real xcar(400),ycar(400),xtrk(400),ytrk(400)
       real trk_time_id(400),car_time_id(400)
     ! variables introduce for the first time in minmax_veh_times
       real tt_st,tt_end,tc_st,tc_end,t_start,t_end
       variables introduce for the first time in log_veh subroutine
       real log_car(400),log_trk(400)
     !
       real xcl(400),ycl(400),xc2(400),yc2(400)
       real xtl(400),ytl(400),xt2(400),yt2(400)
     ! variables introduce for the first time inorder_veh subroutine
       integer n_car_id(400),n_trk_id(400)
       real n_log_car(400),n_log_trk(400)
       real n_trk_time_id(400),n_car_time_id(400)
     ! variables introduce for the first time in direction subroutine
       integer direct
     ! variables introduce for the first time in speed subroutine
       real car_sp(400),trk_sp(400)
     ! variables introduce for the first time in avg_sp subroutine
       real a_sp_cars, a_sp_trks
     ! variables introduce for the first time in cnt_type subroutine
       integer f_type
     ! variables introduce for the first time in x1x2 subroutine
       integer NL
       real dis_atr,dis_x1,dis_x2, Et
       real t0, t1,t2
     ! variable inrtoduced to choose the speed to use to bring vehicles back
     ! to the ATR location
       integer speed_type,ie
     ! variables introduce for the first time in bring_to_atr_x1x2 subroutine
     ! integer c_nx1x2,t_nx1x2
     ! integer x12_car_id(400),x12_trk_id(400)
       integer atr_c_id(400), atr_t_id(400)
       real tminc_x12,tmaxc_x12,tmint_x12,tmaxt_x12
     ! real x12_car_tid(400),x12_trk_tid(400)
     ! real x12_car_log(400),x12_c_sp(400)
     ! real x12_trk_log(400),x12_t_sp(400)
       real atr_c_t(400),atr_t_t(400),atr_c_sp(400),atr_t_sp(400)
     ! variables introduce for the first time in xyATR subroutine
     ! None
     ! variables introduce for the first time in bring_to_atr_a_sp subroutine
       integer cars_in_t,c_sp_i_t,trks_in_t,t_sp_i_t
       real tminc,tmaxc,tmint,tmaxt,a_sp_c_in_t,a_sp_t_in_t
     ! variables introduce for the first time in volume subroutine
     !
       integer trks_in_x1x2,cars_in_x1x2
```

```
      integer cnt_trk_id(400),cnt_car_id(400)
      real  cnt_trk_log(400),cnt_trk_sp(400)
      real  cnt_car_log(400),cnt_car_sp(400)
      real  t1_t,t2_t,t1_c,t2_c
!
! variables introduce for the first time in check_t1t2 subroutine
      integer fail

!
      call centerline(xc,yc,log_cl,numcl,dinc)
      call minmax_cl(numcl,log_cl,cmin,cmax)

      call vehicles(xcar,ycar;xtrk,ytrk,
+          car_id,trk_id,tks,crs,trk_time_id,car_time_id)
! ,tmin,tmax)
      call log_veh(dinc,xc,yc,log_cl,numcl,xcar,ycar,log_car,
+        crs,car_id,car_time_id,xcl,ycl,xc2,yc2)
      call log_veh(dinc,xc,yc,log_cl,numcl,xtrk,ytrk,log_trk,
+          tks,trk_id,trk_time_id,xtl,ytl,xt2,yt2)
!

      call minmax(crs,car_time_id,tc_st,tc_end)
      call minmax(tks,trk_time_id,tt_st,tt_end)
      t_start = tt_st
      t_end  = tt_end
      if(tc_st.lt.tt_st) t_start = tc_st
      if(tc_end.gt.tt_end) t_end = tc_end

!
       call order_veh(log_car,crs,car_id,car_time_id,
+          n_log_car,n_car_id,n_car_time_id)
       call order_veh(log_trk,tks,trk_id,trk_time_id,
+             n_log_trk,n_trk_id,n_trk_time_id)
      This subroutine gets the direction of the Center Line Increase
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      call direction(direct,crs,n_log_car,n_car_id,n_car_time_id)
!  The following subroutine will compute the speeds of vehicles
       print*,'                                          '
       print*,'CARS '
       call speed(crs,n_log_car,n_car_id,n_car_time_id,car_sp,direct)
      call speed(tks,n_log_trk,n_trk_id,n_trk_time_id,trk_sp,direct)
! The following will get me the average speed if trks & and cars
      print*,'                                           '
      call avg_sp(crs,car_sp,a_sp_cars)
      call avg_sp(tks,trk_sp,a_sp_trks)

      print*,'average speed of cars = ', a_sp_cars,'  Kmph'
      print*,'average speed of trucks = ',a_sp_trks,'  Kmph'
      call cnt_type(f_type)
      print*,'type of count is :', f_type
      if(f_type.eq.1)then
      call xyATR(NL,Et,dinc,numcl,xc,yc,log_cl,dis_atr,t0,t1,t2)
      call which_sp(speed_type)
!     print*,' Your start time is = ',t1
!     print*,' Your end time is = ',t2
      if(speed_type.eq.1) then
      call cars
      read*
          call bring_to_atr_a_sp(crs,n_log_car,
```

```
     + n_car_id,n_car_time_id,
     + car_sp,direct,a_sp_cars,dis_atr,
     + atr_c_id,atr_c_t,t1,t2,tminc,tmaxc,
     + cars_in_t,a_sp_c_in_t,c_sp_i_t)
       call trucks
       call bring_to_atr_a_sp(tks,n_log_trk,
     + n_trk_id,n_trk_time_id,
     +  trk_sp,direct,a_sp_trks,dis_atr,
     + atr_t_id,atr_t_t,t1,t2,tmint,tmaxt,
     +  trks_in_t,a_sp_t_in_t,t_sp_i_t)

       else if(speed_type.eq.2) then
       call cars
       read*
        call bring_to_atr(crs,n_log_car,n_car_id,n_car_time_id,
     +   car_sp,direct,a_sp_cars,dis_atr,
     +   atr_c_id,atr_c_t,t1,t2,tminc,tmaxc,
     +   cars_in_t,a_sp_c_in_t,c_sp_i_t)
       read*
       call trucks
       call bring_to_atr(tks,n_log_trk,n_trk_id,n_trk_time_id,
     +    trk_sp,direct,a_sp_trks,dis_atr,
     +   atr_t_id,atr_t_t,t1,t2,tmint,tmaxt,
     +    trks_in_t,a_sp_t_in_t,t_sp_i_t)
       end if
       call out_times(t1,t2,tminc,tmaxc,tmint,tmaxt)
       call check_cl_limits(direct,t1,t2,tminc,tmaxc,tmint,tmaxt,
     +    a_sp_cars,a_sp_trks,dis_atr,
     +    log_cl(2),log_cl(numcl-1),
     +    tc_st,tc_end,tt_st,tt_end)
       call check_t1t2(t1,t2,tminc,tmaxc,tmint,tmaxt,fail)
       if (fail.gt.0) stop
       call comp_par(NL,Et,t1,t2,cars_in_t,trks_in_t,
     +   a_sp_c_in_t,c_sp_i_t,a_sp_t_in_t,t_sp_i_t)

       elseif(f_type.eq.1)then
       call x1x2(NL,Et,dinc,numcl,xc,yc,log_cl,
     +       dis_x1,dis_x2,dis_atr,t0,t1,t2)
       call check_x1x2(dis_x1,dis_x2,direct,log_cl,numcl)
       call veh_in_x1x2(dir,t1,t2,d_x1,d_x2,crs,n_car_id,
           n_log_car,n_car_time_id,car_sp,
           x12_car,x12_idc,x12_lgc,x12_tc,x12_spc)
       call veh_in_x1x2(dir,t1,t2,d_x1,d_x2,tks,n_trk_id,
           n_log_trk,n_trk_time_id,trk_sp,
           x12_trk,x12_idt,x12_lgt,x12_tt,x12_spt)
       call which_sp(speed_type)
       if(speed_type.eq.1)then
       call cars
       call bring_to_atr_x1x2as(crs,n_log_car,n_car_id,
     + n_car_time_id,
     + car_sp,direct,a_sp_cars,dis_atr,dis_x1,dis_x2,
     + tminc_x12,tmaxc_x12,
     + atr_c_id,atr_c_t,atr_c_sp,
     + cars_in_t,a_sp_c_in_t,c_sp_i_t)
       call trucks
       call bring_to_atr_x1x2as(tks,n_log_trk,n_trk_id,
     + n_trk_time_id,
     + trk_sp,direct,a_sp_trks,dis_atr,dis_x1,dis_x2,
     + tmint_x12,tmaxt_x12,
```

```
     + atr_t_id,atr_t_t,atr_t_sp,
     + trks_in_t,a_sp_t_in_t,t_sp_i_t)
       else if(speed_type.eq.2) then
       call cars
       call bring_to_atr_x1x2(crs,n_log_car,n_car_id,
     + n_car_time_id,
     + car_sp,direct,a_sp_cars,dis_atr,dis_x1,dis_x2,
     + tminc_x12,tmaxc_x12,
     + atr_c_id,atr_c_t,atr_c_sp,
     + cars_in_t,a_sp_c_in_t,c_sp_i_t)
       call trucks
       call bring_to_atr_x1x2(tks,n_log_trk,n_trk_id,
     + n_trk_time_id,
     + trk_sp,direct,a_sp_trks,dis_atr,dis_x1,dis_x2,
     + tmint_x12,tmaxt_x12,
     + atr_t_id,atr_t_t,atr_t_sp,
     + trks_in_t,a_sp_t_in_t,t_sp_i_t)
       endif
       call check_t1t2_x1x2(t1,t2, tminc_x12,tmaxc_x12,
     +    tmint_x12,tmaxt_x12)
       end if
 999   stop
       end
```

```
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine centerline(xc,yc,log_cl,k,dinc)
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
implicit none
real xc(800),yc(800),log_cl(800),dist,dinc
integer iof,ior,k,kk,sign
character*20  cent_file
data iof,ior/0,0/
sign = 1
dinc = 0.0
k = 2
print*,'*********************************************************'
print*,'*    Please Enter the Center Line Data File name: '
read(5,10) cent_file
10  format(a)
open(unit =11,file=cent_file,status='old',iostat=iof)
if(iof.ge.0) then
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Reading the mileage of the first point of the CL.
! It is the second point in the array because I am going
! to add a point at the beginning of the CL.  Therefore,
! the first point that we read for the CL is the second
! point of the array.
! This value, the mileage or distance of the first point,
! is the value given on the first line of the CL data file.
read(11,*,iostat=ior) log_cl(2)
print*,'Mileage at first point of CL is ',log_cl(2)
dowhile(ior.ge.0)
read(11,*,iostat=ior)xc(k),yc(k)
if(ior.lt.0)then
 k = k-1
elseif(ior. gt.0)then
 print*,'Error in reading data STOP'
 stop
else
k = k+1
  end if
end do
print*,' Center line points k =', k
close(11)
!!!!!!!!!!!!!!!!!!
 Adding one point at the beginning of the Cener Line
xc(1) = xc(2)-(xc(3)-xc(2))*3
yc(1) = yc(2)-(yc(3)-yc(2))*3
!!!!!!!!!!!!!!!!!!
Adding one point at the end of the Center Line
xc(k+1) = xc(k)+(xc(k)-xc(k-1))*3
yc(k+1) = yc(k)+(yc(k)-yc(k-1))*3
!!!!!!!!!!!!!!!!!!
k = k + 1
!!!!!!!!!!!!!!!!!!!
 At this point if we want to input the mileage at the
! beginning of the CL we can read it in here instead of
! reading it from the data file.
! just use the following 2 lines.
! print*,'Enter the mileage distance at beginning of the CL :'
```

```
       ! read*,log_cl(2)
       !
       ! dist = sqrt((xc(2)-xc(1))**2+(yc(2)-yc(1))**2)
         log_cl(1) = log_cl(2) - dist
         do kk = 3, k
         dist = sqrt((xc(kk)-xc(kk-1))**2+(yc(kk)-yc(kk-1))**2)
         log_cl(kk) = log_cl(kk-1)+dist
         if(kk.gt.2.and.kk.lt.k) then
         if(abs(xc(kk)-xc(kk-1)).gt.dinc) dinc = abs(xc(kk)-xc(kk-1))
         if(abs(yc(kk)-yc(kk-1)).gt.dinc) dinc = abs(yc(kk)-yc(kk-1))
         end if
         end do
         else
         print*,'Center Line Data File failed to open '
         STOP
         end if
         return
         end
         !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
         subroutine  vehicles(xcar,ycar,xtrk,ytrk,
       +      car_id,trk_id,tks,crs,trk_time_id,car_time_id)
!      + ,tmin,tmax)
         !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
         implicit none
         integer  car_id(400),trk_id(400)
         integer iof2,ior,car_trk,tks,crs,pho_n ,classes
         real  trk_time_id(400),car_time_id(400), pho_t,hh,mm,ss
         real xcar(400),ycar(400),xtrk(400),ytrk(400)
         character*20  veh_file
         data iof2,ior/0,0/
         classes = 2
         Print*,'Please Enter the Vehicle Location Data File name:'
         read(5,10) veh_file
      10 format(a)
         open(unit =12,file=veh_file,status='old',iostat=iof2)
        !
         if(iof2.ge.0) then
         ior = 0
         tks = 1
         crs = 1
         dowhile(ior.ge.0)
         read(12,*,iostat=ior) pho_n,hh,mm,ss
         call t_conv(pho_t,hh,mm,ss)
         if(ior.ge.0) then
         car_trk = 1
         do while(car_trk.lt.classes )
         read(12,*)car_trk
         if(car_trk.eq.1) then
         car_id(crs)= 0
         dowhile (car_id(crs).ne.-1)
         read(12,*)car_id(crs),xcar(crs),ycar(crs)
         car_time_id(crs) = pho_t
         if(car_id(crs).ge.0) crs = crs +1
         end do
             else if(car_trk.eq.2) then
         trk_id(tks) = 0
         dowhile(trk_id(tks).ne.-1)
         read(12,*)trk_id(tks),xtrk(tks),ytrk(tks)
         trk_time_id(tks) = pho_t
```

```
      if(trk_id(tks).ge.0) tks = tks+1
      end do
    !  If we have more classes than 2 then we should add an
    !  if statement here and have more arrays to save the data
    !  in them.
      else
      print*,'Error in data format'
      print*,'At photo # : ', pho_n
      print*, car_id(crs-1),xcar(crs-1),ycar(crs-1)
      print*,trk_id(tks-1),xtrk(tks-1),ytrk(tks-1)
      stop
      end if
      end do
      else
      print*,'End of Vehicle data File '
      end if
      end do
      crs = crs - 1
      tks = tks - 1
      print*,'                              '
      print*,' number of cars = ',crs
      print*,' number of trks = ',tks
      print*,'                              '
      else
      print*,'Vehicle Location Data File failed to open'
      STOP
      end if
      return
      end
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine minmax(n,yarray,ymin,ymax)
      implicit none
      integer i,n
      real yarray(n),ymin,ymax
      ymin = yarray(1)              !Set ymin and ymax to
      ymax = yarray(1)              ! first array element.
      do i = 2,n                    !Test balance of array
      if (yarray(i).gt.ymax) then   ! elements.
      ymax = yarray(i)
      elseif (yarray(i).lt.ymin) then
      ymin = yarray(i)
      endif
      enddo
      return
      end                           !End of subroutine.
    !
      subroutine minmax_cl(nc,carray,cmin,cmax)
      implicit none
      integer i,nc
      real carray(nc),cmin,cmax
      cmin = carray(2)              !Set cmin and cmax to
      cmax = carray(2)              !second array element.
      do i = 3,nc                   !Test balance of array
      if (carray(i).gt.cmax) then   !elements.
      cmax = carray(i)
      elseif (carray(i).lt.cmin) then
      cmin = carray(i)
      endif
      enddo
```

```
return
end                              !End of subroutine.
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine t_conv(t_con,hh,mm,ss)
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
implicit none
real hh,mm,ss,t_con
t_con = hh + mm/60. + ss/3600.
return
end
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine t_conv_back(tt_con,hhh,mmm,sss)
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
implicit none
real hhh,mmm,sss,tt_con
hhh = int(tt_con)
mmm = int((tt_con-hhh)*60.)
sss = (((tt_con-hhh)*60)-mmm)*60.
return
end
```

```
      subroutine log_veh(dinc,xc,yc,log_cl,nc,xv,yv,logv,veh,
     +                v_id,v_time_id,x1,y1,x4,y4)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer veh,i,ii,flag,nc
      integer v_id(veh)
      real v_time_id(veh)
      real xc(800),yc(800),log_cl(800),xv(400),yv(400),logv(400)
      real angle,angle1,xv1,yv1,x5,y5,d,dinc,dd
      real x1(400),y1(400),x4(400),y4(400)
      dd = dinc * 2
      do i = 1, veh
      ii= 1
      flag = 1
      dowhile(flag.eq.1)
      if(ii.eq.nc) then
      flag= 0
      else
      if((xc(ii).le.xv(i)+dd.and.xc(ii).ge.xv(i)-dd)
     +  .and.(yc(ii).le.yv(i)+dd.and.yc(ii).ge.yv(i)-dd))then
      angle = atan2d((yc(ii+1)-yc(ii)),(xc(ii+1)-xc(ii)))
      if (angle.gt.360) angle = angle - 360
      angle1 = angle+90.
      xv1=xv(i)+dinc*cosd(angle1)
      yv1=yv(i)+dinc*sind(angle1)
      call intersect(xv1,yv1,xv(i),yv(i),xc(ii),yc(ii),
     +  xc(ii+1),yc(ii+1),x5,y5)
     ! !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
     ! FIRST CHECK.  SEE IF XC1 = XC2 & IF Y5 IS INBETWEEN YC1 & YC2
     ! THEN  CHECK IF YC1 = YC2 & IF X5 IS INBETWEEN XC1 & XC2.
       if((xc(ii).eq.xc(ii+1).and.y5.ge.yc(ii+1).and.
     +    y5.le.yc(ii)) .or.
     +   (xc(ii).eq.xc(ii+1).and.y5.le.yc(ii+1).and.
     +    y5.ge.yc(ii))) then
      call d_log(xc(ii),y5,xc(ii),xc(ii+1),yc(ii),yc(ii+1),
     +    log_cl(ii),log_cl(ii+1),logv(i),x1(i),y1(i))
      x4(i) = xv1
      y4(i) = yv1
       flag = 0
      elseif((yc(ii).eq.yc(ii+1).and.x5.ge.xc(ii+1).and.
     + x5.le.yc(ii)) . or.
     + (yc(ii).eq.yc(ii+1).and.x5.le.xc(ii+1).and.
     + x5.ge.yc(ii)))then
      call d_log(x5,y5,xc(ii),xc(ii+1),yc(ii),yc(ii+1)
     + log_cl(ii),log_cl(ii+1),logv(i),x1(i),y1(i))
      x4(i) = xv1
      y4(i) = yv1
      flag = 0
        else if((x5.ge.xc(ii).and.x5.le.xc(ii+1).and.
     + y5.ge.yc(ii).and.y5.le.yc(ii+1)).or.
     + (x5.ge.xc(ii+1).and.x5.le.xc(ii).and.
     + y5.ge.yc(ii+1).and.y5.le.yc(ii)).or.
     + (x5.ge.xc(ii+1).and.x5.le.xc(ii).and.
     + y5.ge.yc(ii).and.y5.le.yc(ii+1)).or.
     + (x5.ge.xc(ii).and.x5.le.xc(ii+1).and.
     + y5.ge.yc(ii+1).and.y5.le.yc(ii))) then
```

```
        call d_log(x5,y5,xc(ii),xc(ii+1),yc(ii),yc(ii+1)
     +    log_cl(ii),log_cl(ii+1),logv(i),x1(i),y1(i))
        x4(i) = xv1
        y4(i) = yv1
        flag = 0
        else if(
     +  (x5.ge.xc(ii).and.x5.le.(xc(ii)+0.5).and.
     + y5.ge.yc(ii).and.y5.le.(yc(ii)+0.5)).or.
     + (x5.le.xc(ii).and.x5.ge.(xc(ii)+0.5).and.
     + y5.ge.yc(ii).and.y5.le.(yc(ii)+0.5)).or.
     + (x5.ge.xc(ii).and.x5.le.(xc(ii)+0.5).and.
     + y5.le.yc(ii).and.y5.ge.(yc(ii)+0.5)).or.
     + (x5.le.xc(ii).and.x5.ge.(xc(ii)+0.5).and.
     + y5.le.yc(ii).and.y5.ge.(yc(ii)+0.5)).or.
     + (x5.ge.xc(ii+1).and.x5.le.(xc(ii+1)+0.5).and.
     + y5.ge.yc(ii+1).and.y5.le.(yc(ii+1)+0.5)).or.
     +  (x5.le.xc(ii+1).and.x5.ge.(xc(ii+1)+0.5).and.
     + y5.ge.yc(ii+1).and.y5.le.(yc(ii+1)+0.5)).or.
     + (x5.ge.xc(ii+1).and.x5.le.(xc(ii+1)+0.5).and.
     + y5.le.yc(ii+1).and.y5.ge.(yc(ii+1)+0.5)).or.
     + (x5.le.xc(ii+1).and.x5.ge.(xc(ii+1)+0.5).and.
     + y5.le.yc(ii+1).and.y5.ge.(yc(ii+1)+0.5)).or.
     +  (x5.ge.xc(ii).and.x5.le.(xc(ii)-0.5).and.
     + y5.ge.yc(ii).and.y5.le.(yc(ii)-0.5)).or.
     + (x5.le.xc(ii).and.x5.ge.(xc(ii)-0.5).and.
     + y5.ge.yc(ii).and.y5.le.(yc(ii)-0.5)).or.
     + (x5.ge.xc(ii).and.x5.le.(xc(ii)-0.5).and.
     + y5.le.yc(ii).and.y5.ge.(yc(ii)-0.5)).or.
     + (x5.le.xc(ii).and.x5.ge.(xc(ii)-0.5).and.
     + y5.le.yc(ii).and.y5.ge.(yc(ii)-0.5)).or.
     + (x5.ge.xc(ii+1).and.x5.le.(xc(ii+1)-0.5).and.
     + y5.ge.yc(ii+1).and.y5.le.(yc(ii+1)-0.5)).or.
     +  (x5.le.xc(ii+1).and.x5.ge.(xc(ii+1)-0.5).and.
     + y5.ge.yc(ii+1).and.y5.le.(yc(ii+1)-0.5)).or.
     + (x5.ge.xc(ii+1).and.x5.le.(xc(ii+1)-0.5).and.
     + y5.le.yc(ii+1).and.y5.ge.(yc(ii+1)-0.5)).or.
     + (x5.le.xc(ii+1).and.x5.ge.(xc(ii+1)-0.5).and.
     + y5.le.yc(ii+1).and.y5.ge.(yc(ii+1)-0.5)))then
        call d_log(x5,y5,xc(ii),xc(ii+1),yc(ii),yc(ii+1)
            log_cl(ii),log_cl(ii+1),logv(i),x1(i),y1(i))
        x4(i) = xv1
        y4(i) = yv1
        !      flag = 0
        else if ((x5.ge.xc(ii).and.x5.le.(xc(ii)+1).and.
     + y5.ge.yc(ii).and.y5.le.(yc(ii)+1)).or.
     + (x5.le.xc(ii).and.x5.ge.(xc(ii)+1).and.
     + y5.ge.yc(ii).and.y5.le.(yc(ii)+1)).or.
     + (x5.ge.xc(ii).and.x5.le.(xc(ii)+1).and.
     + y5.le.yc(ii).and.y5.ge.(yc(ii)+1)).or.
     + (x5.le.xc(ii).and.x5.ge.(xc(ii)+1).and.
     + y5.le.yc(ii).and.y5.ge.(yc(ii)+1)).or.
     + (x5.ge.xc(ii+1).and.x5.le.(xc(ii+1)+1).and.
     + y5.ge.yc(ii+1).and.y5.le.(yc(ii+1)+1)).or.
     + (x5.le.xc(ii+1).and.x5.ge.(xc(ii+1)+1).and.
     + y5.ge.yc(ii+1).and.y5.le.(yc(ii+1)+1)).or.
     + (x5.ge.xc(ii+1).and.x5.le.(xc(ii+1)+1).and.
     + y5.le.yc(ii+1).and.y5.ge.(yc(ii+1)+1)).or.
```

```
     + (x5.le.xc(ii+1).and.x5.ge.(xc(ii+1)+1).and.
     + y5.le.yc(ii+1).and.y5.ge.(yc(ii+1)+1)).or.
     +  (x5.ge.xc(ii).and.x5.le.(xc(ii)-1).and.
     + y5.ge.yc(ii).and.y5.le.(yc(ii)-1)).or.
     + (x5.le.xc(ii).and.x5.ge.(xc(ii)-1).and.
     + y5.ge.yc(ii).and.y5.le.(yc(ii)-1)).or.
     + (x5.ge.xc(ii).and.x5.le.(xc(ii)-1).and.
     + y5.le.yc(ii).and.y5.ge.(yc(ii)-1)).or.
     + (x5.le.xc(ii).and.x5.ge.(xc(ii)-1).and.
     + y5.le.yc(ii).and.y5.ge.(yc(ii)-1)).or.
     + (x5.ge.xc(ii+1).and.x5.le.(xc(ii+1)-1).and.
     + y5.ge.yc(ii+1).and.y5.le.(yc(ii+1)-1)).or.
     + (x5.le.xc(ii+1).and.x5.ge.(xc(ii+1)-1).and.
     + y5.ge.yc(ii+1).and.y5.le.(yc(ii+1)-1)).or.
     + (x5.ge.xc(ii+1).and.x5.le.(xc(ii+1)-1).and.
     + y5.le.yc(ii+1).and.y5.ge.(yc(ii+1)-1)).or.
     + (x5.le.xc(ii+1).and.x5.ge.(xc(ii+1)-1).and.
     + y5.le.yc(ii+1).and.y5.ge.(yc(ii+1)-1)))then
       call d_log(x5,y5,xc(ii),xc(ii+1),yc(ii),yc(ii+1)
     +   log_cl(ii),log_cl(ii+1),logv(i),x1(i),y1(i))
       x4(i) = xv1
       y4(i) = yv1
       else if ((x5.ge.xc(ii).and.x5.le.(xc(ii)+5).and.
     + y5.ge.yc(ii).and.y5.le.(yc(ii)+5)).or.
     + (x5.le.xc(ii).and.x5.ge.(xc(ii)+5).and.
     + y5.ge.yc(ii).and.y5.le.(yc(ii)+5)).or.
     + (x5.ge.xc(ii).and.x5.le.(xc(ii)+5).and.
     + y5.le.yc(ii).and.y5.ge.(yc(ii)+5)).or.
     + (x5.le.xc(ii).and.x5.ge.(xc(ii)+5).and.
     + y5.le.yc(ii).and.y5.ge.(yc(ii)+5)).or.
     + (x5.ge.xc(ii+1).and.x5.le.(xc(ii+1)+5).and.
     + y5.ge.yc(ii+1).and.y5.le.(yc(ii+1)+5)).or.
     + (x5.le.xc(ii+1).and.x5.ge.(xc(ii+1)+5).and.
     + y5.ge.yc(ii+1).and.y5.le.(yc(ii+1)+5)).or.
     + (x5.ge.xc(ii+1).and.x5.le.(xc(ii+1)+5).and.
     + y5.le.yc(ii+1).and.y5.ge.(yc(ii+1)+5)).or.
     + (x5.le.xc(ii+1).and.x5.ge.(xc(ii+1)+5).and.
     + y5.le.yc(ii+1).and.y5.ge.(yc(ii+1)+5)).or.
     + (x5.ge.xc(ii).and.x5.le.(xc(ii)-5).and.
     + y5.ge.yc(ii).and.y5.le.(yc(ii)-5)).or.
     + (x5.le.xc(ii).and.x5.ge.(xc(ii)-5).and.
     + y5.ge.yc(ii).and.y5.le.(yc(ii)-5)).or.
     + (x5.ge.xc(ii).and.x5.le.(xc(ii)-5).and.
     + y5.le.yc(ii).and.y5.ge.(yc(ii)-5)).or.
     + (x5.le.xc(ii).and.x5.ge.(xc(ii)-5).and.
     + y5.le.yc(ii).and.y5.ge.(yc(ii)-5)).or.
     + (x5.ge.xc(ii+1).and.x5.le.(xc(ii+1)-5).and.
     + y5.ge.yc(ii+1).and.y5.le.(yc(ii+1)-5)).or.
     + (x5.le.xc(ii+1).and.x5.ge.(xc(ii+1)-5).and.
     + y5.ge.yc(ii+1).and.y5.le.(yc(ii+1)-5)).or.
     + (x5.ge.xc(ii+1).and.x5.le.(xc(ii+1)-5).and.
     + y5.le.yc(ii+1).and.y5.ge.(yc(ii+1)-5)).or.
     + (x5.le.xc(ii+1).and.x5.ge.(xc(ii+1)-5).and.
     + y5.le.yc(ii+1).and.y5.ge.(yc(ii+1)-5)))then
       call d_log(x5,y5,xc(ii),xc(ii+1),yc(ii),yc(ii+1)
     +      log_cl(ii),log_cl(ii+1),logv(i),x1(i),y1(i))
       x4(i) = xv1
       y4(i) = yv1
       end if
```

```
end if
ii = ii+1
end if
end do
end do
return
end
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine intersect(x1,y1,x2,y2,x3,y3,x4,y4,x5,y5)
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
implicit none
real x1,x2,x3,x4,x5,y1,y2,y3,y4,y5
real  a1,b1,c1,a2,b2,c2
a1 = y2-y1
b1 = x1 - x2
c1 = x2*y1 - x1*y2
a2 = y4-y3
b2 = x3 -x4
c2 = x4*y3 -x3*y4
x5 = (b1*c2-b2*c1)/(a1*b2-a2*b1)
y5 = (c1*a2 - c2*a1)/(a1*b2-a2*b1)
return
end
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine d_log(x5,y5,xc1,xc2,yc1,yc2,logc1,logc2,logv,xl,yl)
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
implicit none
real x5,y5,xc1,yc1,xc2,yc2,logc1,logc2,logv,xl,yl,d
xl = x5
yl = y5
if(logc1.gt.logc2)then
d = ((x5-xc2)**2+(y5-yc2)**2)**.5
logv = d + logc2
else
d = ((x5-xc1)**2+(y5-yc1)**2)**.5
logv = d + logc1
end if
return
end
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine dis_xy(x5,y5,xc1,xc2,yc1,yc2,logc1,logc2,logv)
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
implicit none
real x5,y5,xc1,yc1,xc2,yc2,logc1,logc2,logv,d
if(logc1.gt.logc2)then
d = ((x5-xc2)**2+(y5-yc2)**2)**.5
logv = d + logc2
else
d = ((x5-xc1)**2+(y5-yc1)**2)**.5
logv = d + logc1
end if
return
end
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine order_veh(log_veh,veh,veh_id,veh_time_id,
+          n_log_veh,n_veh_id,n_veh_time_id)
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
implicit none
integer iii,ii,i, veh,veh_id(400),n_veh_id(400)
```

```
      integer min_veh_id,max_veh_id
      integer flag,flag2,latestveh,veh_del(400)
      real  veh_time_id(400),n_veh_time_id(400)
      real log_veh(400),n_log_veh(400)

100 format(/,60('-'),/,
    +  ' ID     DistanceLog     Time   '/60('-'),/,
    +     i9/60('-'))
      do i= 1,veh
      veh_del(i) = 0
      end do

    ! This loop will organize the data in the vehicle ID assending order format
    ! that we want
    !
      call minmax(veh,veh_id,min_veh_id,max_veh_id)
      print*,'                                          '
      print*,'minimum vehicle ID is = ',min_veh_id
      print*,'maximum vehicle ID is = ',max_veh_id
      print*,'                                          '
      latestveh = min_veh_id
      i = 1
      dowhile(i.le.veh.and.latestveh.le.max_veh_id)
      flag = 0
      ii = 1
      do while(flag.eq.0.and.ii.le.veh)
      if(veh_id(ii).eq.latestveh.and.veh_del(ii).eq.0) then
      n_veh_id(i) = latestveh
      n_log_veh(i)= log_veh(ii)
      n_veh_time_id(i) = veh_time_id(ii)
      veh_del(ii) = 1
      flag = 1
      latestveh = latestveh + 1
      i = i+1
    !
      else if(ii.eq.veh) then
    !
       latestveh=latestveh + 1
      end if
       ii= ii+1
      end do
      flag2 = 0
      iii = ii
      do while(flag2.eq.0.and.iii.le.veh)
      if(veh_id(iii).eq.latestveh-1.and.veh_del(iii).ne.1.
    +        and.iii.le.veh)then
       flag2= 1
       latestveh = latestveh - 1
      end if
      iii = iii+1
      if(flag.eq.0) latestveh = latestveh + 1
      end do
      end do
      return
      end
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine speed(veh,n_veh_lg,n_veh_id,n_veh_time,veh_sp,direct)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
```

```
      integer veh,i, direct
      integer n_veh_id(400)
      real veh_sp(400),n_veh_lg(400),n_veh_time(400)
      real dt_time
!  Now We request the time difference between the set of photos.
      write(6,300)
      veh_sp(1) = 0
      do i= 1, veh-1
      if(n_veh_id(i).eq.n_veh_id(i+1)) then
      dt_time = (n_veh_time(i+1)-n_veh_time(i))
      veh_sp(i+1)=(n_veh_lg(i+1)-n_veh_lg(i))/dt_time/1000*direct
!     *3.6)*direct
      if(veh_sp(i+1).lt.40) veh_sp(i+1)= 0.
      if(veh_sp(i+1).gt.150) veh_sp(i+1)= 0.
      else
      veh_sp(i+1)= 0
      end if
       end do
      return
      end
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine avg_sp(n,veh_sp,a_sp_vehs)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer n,i,nn
      real veh_sp(n),a_sp_vehs,sum
      sum = 0
      nn = 0
      do i=1,n
      if(veh_sp(i).ne.0) then
       nn = nn +1
      sum = sum + veh_sp(i)
      end if
      end do
      a_sp_vehs = sum /nn
      return
      end
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine direction(direct,veh,n_log_veh,n_veh_id,n_veh_time)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer direct ,veh,n_veh_id(veh),i, n
      real n_log_veh(veh),n_veh_time(veh),sp
      n = 0
      direct = 1
      do i= 1, veh/2
      if(n_veh_id(i).eq.n_veh_id(i+1)) then
       sp=(n_log_veh(i+1)-n_log_veh(i))/1000/
     + (n_veh_time(i+1)-n_veh_time(i))
        if(sp.lt.0) n = n-1
      if(sp.gt.0) n = n+1
        end if
      end do
      if(n.lt.0)direct = -1
!  print*,'
!  print*,'n is =  ', n
!  print*,'Direction is = ', direct
!  print*,'
       return
```

352	end
353	!!!!+!!

```
      subroutine xyATR(NL,Et,dinc,nc,xc,yc,log_cl,dis_atr,t0,t1,t2)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer NL,nc , iof,ior
      integer  p_id(2)
      real xc(nc),yc(nc),log_cl(nc),xp(1),yp(1),log_p(1)
      real xp1(1),yp1(1),xp4(1),yp4(1),p_time(1),dinc
      real hh,mm,ss,t0,t1,hh1,mm1,ss1,t2,hh2,mm2,ss2
      real dis_atr, Et
      character*20  xy_atr
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    ! ASK FOR A FILE WITH XY-ATR AND t0 DATA FILE AND FIND THE
    ! COORDINATES WITH REFERNCE TO THE CENTER LINE.

      print*,'Please Enter the name of the file that has '
      print*,'XY-ATR,to, t1, t2 '
      read(5,10) xy_atr
   10 format(a)
      open(unit =9,file=xy_atr,status='old',iostat=iof)
      if(iof.ge.0) then
      read(9,*,iostat=ior)NL
      read(9,*,iostat=ior)Et
      read(9,*,iostat=ior)xp(1),yp(1),hh,mm,ss,
     +    mm1,ss1,HI4,mm2,ss2
      if(ior.ne.0)then
      print*,'Check data format in data file???'
      stop
      end if
      call t_conv(t0,hh,mm,ss)
      call t_conv(t1,hh1,mm1,ss1)
      call t_conv(t2,hh2,mm2,ss2)
      p_time(1) = t0
      p_id(1) = 1
      call log_location(dinc,xc,yc,log_cl,nc,xp,yp,log_p,1)
    !
      dis_atr = log_p(1)
      print*,'
      print101, dis_atr,hh,mm,ss,hh1,mm1,ss1,t1,hh2,mm2,ss2,t2
  101  format(' ATR is at distance : ',f10.2/
     +      ' Base time is :',f4.0,f4.0,f6.3/
     +      ' Start time is :',f4.0,f4.0,f6.3,f11.5/
     +      ' End time  is :',f4.0,f4.0,f6.3,f11.5)
      else
      print*,'X1,X2,t data file failed to open '
      STOP
      end if
      return
      end
    ! !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !  NOTES
    !  This Subroutine  brings all the vehicles to the atr location
    !  using the average speed of all the vehicles in the same
    !  class to compute the time by taking distance/speed
    ! !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine bring_to_atr_a_sp(veh,n_log_veh,
     +    n_veh_id,n_veh_time_id,
```

```
     +      veh_sp,dir,a_sp_veh,log_atr,atr_v_id,atr_t_veh,
     +      t1,t2,tminv,tmaxv,iii,a_sp_in,i_sp)
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer dir,veh,k,i,i1,i2,flag1,flag2,flag3,iii,i_sp
      integer n_veh_id(veh), atr_v_id(400),casev(400)
      integer id_in_t(400)
      real  sp_in_t(400)
      real  n_log_veh(veh),veh_sp(veh),n_veh_time_id(veh)
      real atr_t_veh(veh),atr_v_sp(400)
      real a_sp_veh,spd
      real log_atr,t1,t2,tminv,tmaxv
      real l_i1, l_i2
      real tmin,tmin_diff,t_diff,t_diff2,tt
      real a_sp_in,sum_sp
      a_sp_in = 0
      i_sp = 0
      tminv = 9999
      tmaxv =  0
      print13,log_atr,t1,t2
   13  format(60('-'),/'ATR dist inside the subroutine is:',f9.1,
     + /'Start time for count is = ',f11.7,
     + /'End time for count is   = ',f11.7,
     + //'The values of the log and speed are:'//
     + '#     ID            Dist_veh       Speed'/60('-'))

       do i=1,veh
        print*,i,n_veh_id(i),n_log_veh(i),veh_sp(i)
       end do
       tmin = 1./(30.*3600.)
       tmin_diff = 5./3600.
      flag1 = 0
      i = 0
      i1 = 1
      i2 = 1
      spd = a_sp_veh
      do while(flag1.eq.0)
      i = i + 1
      flag2 = 0
      do while(flag2.eq.0)
      if(n_veh_id(i1).eq.n_veh_id(i2+1))then
      i2= i2+1
      else
      flag2= 2
      end if
      end do
      atr_v_id(i) = n_veh_id(i1)
      if(i1.eq.i2) then
      casev(i) = 1
      atr_t_veh(i) = n_veh_time_id(i1) +
     +          (log_atr - n_log_veh(J4))/(1000*spd)* dir
      atr_v_sp(i) = 0.0
        elseif(i2.gt.i1) then
      flag3 = 0
      do k = i1,i2-1
      if((n_log_veh(k).ge.log_atr.and.
     +    n_log_veh(k+1).le.log_atr).or.
     +   (n_log_veh(k).le.log_atr.and.
     +    n_log_veh(k+1).ge.log_atr))then
```

```
        atr_t_veh(i) = n_veh_time_id(k) +
     +      (log_atr - n_log_veh(k))/(1000*spd)*dir
        atr_v_sp(i) = veh_sp(k+1)
        flag3 = 1
            casev(i) = 2
        end if
        end do
        if(flag3.eq.0) then
        l_i1 = n_log_veh(i1)
        l_i2 = n_log_veh(i2)
        if (dir.gt.0) then
        if(l_i2.le.log_atr) then
        atr_t_veh(i) = n_veh_time_id(i2) +
     +  (log_atr - n_log_veh(i2))/(1000*spd)*dir
        atr_v_sp(i) = veh_sp(i2)
           casev(i) = 3
        elseif(l_i1.ge.log_atr) then
        atr_t_veh(i) = n_veh_time_id(i1) +
     +   (log_atr - n_log_veh(i1))/(1000*spd)*dir
        atr_v_sp(i) = veh_sp(i1+1)
           casev(i) = 4
        end if
        else if(dir.lt.0)then
        if(l_i1.le.log_atr) then
        atr_t_veh(i) = n_veh_time_id(i1) +
     +   (log_atr - n_log_veh(i1))/(1000*spd)*dir
        atr_v_sp(i) = veh_sp(i1+1)
           casev(i) = 5
        elseif(l_i2.ge.log_atr) then
         atr_t_veh(i) = n_veh_time_id(i2) +
     +          (log_atr - n_log_veh(J5))/(1000*spd)*dir
        atr_v_sp(i) = veh_sp(i2)
        casev(i) = 6
        end if
        end if
        end if
        else
        print*,'
        print*,'** SOME THING IS WRONG IN THE CHECK   **'
        print*,'** AT VEHICLE #',i,'          **'
        print*,'
        end if
        if(atr_t_veh(i).gt.tmaxv) tmaxv = atr_t_veh(i)
        if(atr_t_veh(i).lt.tminv) tminv = atr_t_veh(i)
        if(i2.eq.veh) flag1 = 10
        i1 = i2+1
        i2 = i1
        end do
        print*,
        print*,'** Direction is   = ', dir,'      **'
        print*,'** # of vehicles = ', i , '      **'
        print*,
        do k = 1,i
        write(6,130)k,atr_v_id(k),atr_t_veh(k),atr_v_sp(k),casev(k)
        end do
  130 format(i5, i7,3x,f12.7,f9.3,i4)
        iii = 0
        do k = 1,i
        if(atr_t_veh(k).ge.t1.and.atr_t_veh(k).le.t2) then
```

```
      iii = iii + 1
      id_in_t(iii) = atr_v_id(k)
      sp_in_t(iii) = atr_v_sp(k)
      end if
      end do
      print140,tminv, tmaxv
140   format(60('-')/'For this segment Tmin is = ',f11.7/
     '         &     Tmax is = ',f11.7//
        60('-')/'Vehicles that are included in the count are',
       /60('-')/'  #    ID    SPEED(kmph)   '/60('-'))
      i_sp = 0
      sum_sp = 0
      do k = 1,iii
      print150,k,id_in_t(k),sp_in_t(k)
      if(sp_in_t(k).gt.0) then
       i_sp = i_sp + 1
      sum_sp = sum_sp + sp_in_t(k)
      end if
      end do
      if(i_sp.ne.0)  a_sp_in = sum_sp / i_sp
      print160,a_sp_in, i_sp
160   format(60('-')/'The average speed of these vehicles is:',
     +   f9.4,' kmph ',/'This is for',i4,' Vehicles'/60('-')/)
150    format(2(i5),f11.4)
      return
      end
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
       NOTES
       This Subroutine is based on the fact the same vehilce that
       sppears more than once is organized in the format that the
       the vehicle's first appearance is listed firts.

      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine bring_to_atr(veh,n_log_veh,n_veh_id,n_veh_time_id,
     +      veh_sp,dir,a_sp_veh,log_atr,atr_v_id,atr_t_veh,
     +     t1,t2,tminv,tmaxv,iii,a_sp_in,i_sp)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer dir,veh,k,i,i1,i2,flag1,flag2,flag3,iii,i_sp
      integer n_veh_id(veh), atr_v_id(400),casev(400)
      integer id_in_t(400)
      real  sp_in_t(400)
      real  n_log_veh(veh),veh_sp(veh),n_veh_time_id(veh)
      real atr_t_veh(veh),atr_v_sp(400)
      real a_sp_veh,spd
       real log_atr,t1,t2,tminv,tmaxv
      real l_i1, l_i2
       real tmin,tmin_diff,t_diff,t_diff2,tt
      real a_sp_in,sum_sp
      i_sp = 0
      spd = 0
      tminv = 99999
      tmaxv = 0
      print13,log_atr,t1,t2

13    format(60('-'),/'ATR dist inside the subroutine is:',f9.1,
     +  /'Start time for count is = ',f11.7,
     +  /'End time for count is   = ',f11.7,
     +  //'The values of the log and speed are:'//
```

```
     + '#     ID           Dist_veh       Speed'/60('-'))

        do i=1,veh
          print*,i,n_veh_id(i),n_log_veh(i),veh_sp(i)
         end do
        tmin = 1./(30.*3600.)
        tmin_diff = 5./3600.
       flag1 = 0
       i = 0
       i1 = 1
       i2 = 1
       do while(flag1.eq.0)
       i = i + 1
       flag2 = 0
       do while(flag2.eq.0)
       if(n_veh_id(i1).eq.n_veh_id(i2+1))then
       i2= i2+1
       else
       flag2= 2
       end if
       end do
       atr_v_id(i) = n_veh_id(i1)
       if(i1.eq.i2) then
       casev(i) = 1
       atr_t_veh(i) = n_veh_time_id(i1) +
     +      (log_atr - n_log_veh(J4))/(1000*a_sp_veh)* dir
       atr_v_sp(i) = 0.0
         elseif(i2.gt.i1) then
       flag3 = 0
       do k = i1,i2-1
       if((n_log_veh(k).ge.log_atr.and.
     +     n_log_veh(k+1).le.log_atr).or.
     +    (n_log_veh(k).le.log_atr.and.
     +     n_log_veh(k+1).ge.log_atr))then
       spd = veh_sp(k+1)
       if(veh_sp(k+1).eq.0) spd = a_sp_veh
       atr_t_veh(i) = n_veh_time_id(k) +
     +          (log_atr - n_log_veh(k))/(1000*spd)*dir
       atr_v_sp(i) = veh_sp(k+1)
       flag3 = 1
       casev(i) = 2
       end if
       end do
       if(flag3.eq.0) then
       l_i1 = n_log_veh(i1)
       l_i2 = n_log_veh(i2)
       if (dir.gt.0) then
       if(l_i2.le.log_atr) then
       spd = veh_sp(i2)
       if(veh_sp(i2).eq.0) spd = a_sp_veh
       atr_t_veh(i) = n_veh_time_id(i2) +
     +   (log_atr - n_log_veh(i2))/(1000*spd)*dir
       atr_v_sp(i) = veh_sp(i2)
       casev(i) = 3
       elseif(l_i1.ge.log_atr) then
       spd = veh_sp(i1+1)
       if(veh_sp(i1+1).eq.0) spd = a_sp_veh
       atr_t_veh(i) = n_veh_time_id(i1) +
     +  (log_atr - n_log_veh(i1))/(1000*spd)*dir
```

```
      atr_v_sp(i) = veh_sp(i1+1)
      casev(i) = 4
      end if
      else if(dir.lt.0)then
      if(l_i1.le.log_atr) then
       spd = veh_sp(i1+1)
       if(veh_sp(i1+1).eq.0) spd = a_sp_veh
       atr_t_veh(i) = n_veh_time_id(i1) +
     +   (log_atr - n_log_veh(i1))/(1000*spd)*dir
      atr_v_sp(i) = veh_sp(i1+1)
      casev(i) = 5
      elseif(l_i2.ge.log_atr) then
       spd = veh_sp(i2)
       if(veh_sp(i2).eq.0) spd = a_sp_veh
       atr_t_veh(i) = n_veh_time_id(i2) +
     +        (log_atr - n_log_veh(J5))/(1000*spd)*dir
      atr_v_sp(i) = veh_sp(i2)
      casev(i) = 6
      end if
      end if
      end if
      else
      print*,'*******************************************'
      print*,'**  SOME THING IS WRONG IN THE CHECK   **'
      print*,'**  AT VEHICLE # ',i,'         **'
      print*,'*******************************************'
      end if
      if(atr_t_veh(i).gt.tmaxv) tmaxv = atr_t_veh(i)
      if(atr_t_veh(i).lt.tminv) tminv = atr_t_veh(i)
      if(i2.eq.veh) flag1 = 10
      i1 = i2+1
      i2 = i1
      end do
      print*,'*******************************************'
      print*,'**  Direction is   = ', dir,'     **'
      print*,'**  # of vehicles = ', i , '     **'
      print*,'*******************************************'
      do k = 1,i
      write(6,130)k,atr_v_id(k),atr_t_veh(k),atr_v_sp(k),casev(k)
      end do
 130   format(i5, i7,3x,f12.7,f9.3,i4)
      iii = 0
      do k = 1,i
      if(atr_t_veh(k).ge.t1.and.atr_t_veh(k).le.t2) then
      iii = iii + 1
      id_in_t(iii) = atr_v_id(k)
      sp_in_t(iii) = atr_v_sp(k)
      end if
       end do
      print140,tminv, tmaxv
 140  format(60('-')/'For this segment Tmin is = ',f11.7/
     +  '         &    Tmax is = ',f11.7//
     +   60('-')/'Vehicles that are included in the count are',
     +    /60('-')/'  #   ID    SPEED(kmph)   '/60('-'))
      i_sp = 0
      sum_sp = 0
      do k = 1,iii
      print150,k,id_in_t(k),sp_in_t(k)
      if(sp_in_t(k).gt.0) then
```

```
      i_sp = i_sp + 1
      sum_sp = sum_sp + sp_in_t(k)
      end if
      end do
      a_sp_in = sum_sp / i_sp
      print160,a_sp_in, i_sp
160   format(60('-')/'The average speed of these vehicles is:',
     +   f9.4,' kmph ',/'This is for',i4,' Vehicles'/60('-')/)
150   format(2(i5),f11.4)
      print*,'Cars with Speed = ',a_sp_in ,' Speed=',i_sp
      return
      end
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      NOTES
      This Subroutine computes that earlist time and the latest
       time that we can do the count in.  Based on the first and last
       points of the CL and using the speeds of the cars and the speeds
       of the trucks.

      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine t_cl_b_e(d_strt,d_end,dirr,tmin,tmax,
     + +              a_sp_cars,a_sp_trks,dis_atr,t1,t2)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer dirr
      real d_strt,d_end,dir,tmin,tmax
      real a_sp_cars,a_sp_trks,dis_atr,t1,t2
      real car_strt_t,car_end_t,trk_strt_t,trk_end_t
      print*,'Inside the subroutine '
      print*,' Your start time is = ',t1
      print*,' Your end time is = ',t2
      if(dirr.gt.0)then
      car_strt_t = tmax+(dis_atr - d_end)/ (1000*a_sp_cars)
      car_end_t  = tmin+(dis_atr - d_strt)/(1000*a_sp_cars)
      trk_strt_t = tmax+(dis_atr - d_end)/ (1000*a_sp_trks)
      trk_end_t  = tmin+(dis_atr - d_strt)/(1000*a_sp_trks)
       else if(dirr.lt.0) then
         car_strt_t = tmax-(dis_atr - d_strt)/ (1000*a_sp_cars)
         car_end_t  = tmin-(dis_atr - d_end)/(1000*a_sp_cars)
         trk_strt_t = tmax-(dis_atr - d_strt)/ (1000*a_sp_trks)
         trk_end_t  = tmin-(dis_atr - d_end)/(1000*a_sp_trks)

       end if
      print205,d_strt,d_end,dis_atr,tmin,tmax,dirr,
     +      a_sp_cars,a_sp_trks
      print210,car_strt_t,car_end_t,trk_strt_t,trk_end_t,t1,t2
      if(t1.lt.car_strt_t.or.t1.lt.trk_strt_t) then
      print*,'Your start time is invalid '
      print*,'TErminating process '
      stop
      end if
      if(t2.gt.car_end_t.or.t2.gt.trk_end_t) then
      print*,'Your end time is invalig '
      print*,'TErminating process '
      stop
      end if
 205  format(/,70('-')/
     +   ' Start distance of CL is =',f11.2/
     +   ' End distance of CL is =',f11.2/
```

```
     +    ' Distance of ATR is = ',f11.2/
     +    ' Start time of photos is =',f11.7/
     +    ' End time of photos is =',f11.7/
     +    ' Direction is  =',i4 /
     +    ' Average speed of cars is =',f10.2/
     +     'Average speed of Trucks is =',f10.2/)
 210  format(/,70('-')/
     +    ' Start time of Car count =',f11.7/
     +    ' End time of Car count   =',f11.7/
     +    ' Start time of Truck count =',f11.7/
     +    ' End time of Truck count   =',f11.7//
     +    ' Your start time is = ',f11.7/
     +    ' Your end time is = ',f11.7/70('-')/)
      return
      end
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
      subroutine  comp_par(NL,Et, t1,t2,vol_c,vol_t,
     +   a_sp_c,c_i,a_sp_t,t_i)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer vol_c,vol_t,c_i,t_i,tot_vol
      integer NL
      real t1,t2,t_interv,h1,m1,s1
      real  a_sp_c,a_sp_t,s_m_sp
      real Et, per_t,fhv, flow_tot,eq_pc_fl,density,dens_pc_ln
      print*,'cars with speed = ',c_i,',  Their speed=',a_sp_c
      print*,'trucks with speed = ',t_i,',  Their speed=',a_sp_t
      print*,'Number of lanes = ',NL
      print*,'The value of Et read from data file :',Et
     ! Et = 1.5
      tot_vol = vol_c + vol_t
      s_m_sp = (a_sp_c * c_i + a_sp_t * t_i)/((c_i+t_i)*1.)/1.609
      t_interv = (t2 - t1)*60
      per_t = vol_t*1. /tot_vol* 100.
      fhv = 1/(1+per_t/100*(Et-1))
      flow_tot = tot_vol * 60 /t_interv
      eq_pc_fl = flow_tot /fhv/NL
      density =  flow_tot / s_m_sp
      dens_pc_ln = density / NL/fhv
      call t_conv_back(t1,h1,m1,s1)
      print500,h1,m1,s1,t1,t_interv,vol_c,vol_t,tot_vol,per_t,
     +  eq_pc_fl,s_m_sp,density,dens_pc_ln
  500 format(//60('-')/
     +  ' Measure/ Parameter [units]      '/60('-')/
     +  ' Initila Clock Time to [hh:mm:ss]',11x,f3.0,f3.0,f6.2,f10.6/
     +  ' Time interval dt [mins]         ',11x,f7.2/
     +  ' Volume of Cars Vc [veh]         ',11x,i6/
     +  ' Volume of Trucks Vt [veh]       ',11x,i6/
     +  ' Total Volume Vveh  [veh]        ',11x,i6/
     +  ' Percent Trucks Ptrk  [%]        ',11x,f7.2/
     +  ' Equiv Passenger Car Flow Qpcpl[pcphpc]',5x,f7.2/
     +  ' Space Mean Speed Us  [mph]      ',11x,f7.2/
     +  ' Density K  [veh/mi]             ',11x,f7.2/
     +  ' Equiv Passenger Car Density Kpcpl[pcpmpl]',4x,f7.2/
     +  60('-'))
      return
      end
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
      subroutine x1x2(NL,Et,dinc,nc,xc,yc,log_cl,
     +    dis_x1,dis_x2,dis_atr,t0,tx1,tx2)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

      implicit none
      integer nc , iof,ior,i
      integer  NL
      real xc(nc),yc(nc),log_cl(nc),xp(3),yp(3),log_p(3)
      real Et, dinc
      real h(3),m(3),s(3),t0,tx1,tx2,dis_x1, dis_x2,dis_atr
      character*20  x1x2_file
!     The following variables can be deleted if we delete the variables
!     in the subroutine log_veh
!     These are being used here only because I have to pass then to
!     this Subroutine.
      real xp1(3),yp1(3),xp4(3),yp4(3)
!     !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!      ASK FOR A FILE WITH X1 & X2 POINTS AND THEN FIND THESE POINTS
!      WITH REFERENCE TO THE CENTER LINE.
      iof = 0
      print*,'Please Enter the name of the file that has '
      print*,'X1,X2 points'
      read(5,10) x1x2_file
10      format(a)
      open(unit =9,file=x1x2_file,status='old',iostat=iof)
      if(iof.ge.0) then
      read(9,*,iostat=ior) NL
      read(9,*,iostat=ior) Et
      do i = 1, 3
!       only the value of Xp and Yp are needed for the ones in the
!       report.  But for the later more comprehensive one these values
!       are needed.
      read(9,*,iostat=ior)xp(i),yp(i),h(i),m(i),s(i)
      if(ior.ne.0)then
      print*,'Check data format in data file???'
      stop
      end if
      end do
        call t_conv(t0,h(1),m(1),s(1))
        call t_conv(tx1,h(2),m(2),s(2))
       call t_conv(tx2,h(3),m(3),s(3))
      print*,'t0=', t0,'   xt1 =',tx1,'   xt2 =',tx2
      call log_location(dinc,xc,yc,log_cl,nc,xp,yp,log_p,3)
      dis_atr = log_p(1)
      dis_x1 = log_p(2)
      dis_x2 = log_p(3)
      print*,'————————————————————————————'
      print*,' x1 and x2 points are: '
      print*,log_p(2),'   &  ',log_p(3)
      print*,'  ATR at :',log_p(1)
      print*,'                                    '
      else
      print*,'X1,X2,t data file failed to open '
      stop
      end if
      return
```

```
      end
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!     NOTES
!     This Subroutine is based on the fact the same vehilce that
!     appears more than once is organized in the format that the
!     the vehicle's first appearance is listed firts.
!     This subroutine is only for the vehicles that are between
!     the X1-X2 locations.
!     This subroutine brings all the vehicles back to the ATR
!     using the individual speeds of vehicles.
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine bring_to_atr_x1x2as(x12_veh,x12_lgv,x12_idv,
     +        x12_tv,x12_spv,
     +        dir,a_sp_veh,x_atr,x1,x2,
     +        tminv_x1x2,tmaxv_x1x2,
     +       atr_v_id,atr_t_veh,atr_v_sp,
     +        i,a_sp_in,i_sp)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer dir,x12_veh,k,i,i1,i2,flag1,flag2,flag3,iii,i_sp
      integer x12_idv(x12_veh), atr_v_id(400),casev(400)
      integer id_in_t(400)
      real sp_in_t(400)
      real x12_lgv(x12_veh),x12_spv(x12_veh),x12_tv(x12_veh)
      real atr_t_veh(x12_veh),atr_v_sp(400)
      real a_sp_veh,spd
      real x_atr,x1,x2,tminv_x1x2,tmaxv_x1x2
      real l_i1, l_i2
      real tmin,tmin_diff,t_diff,t_diff2,tt
      real a_sp_in,sum_sp
      real log_min,log_max
      i_sp = 0
      spd = 0
      tminv_x1x2 = 9999.9999
      tmaxv_x1x2 = 0
      print13,x_atr,x1,x2
   13 format(60('-'),/'ATR dist inside the subroutine is:',f9.1,
     +     /'Start distance for count is = ',f11.5,
     +     /'End  distance for count is  = ',f11.5,
     +    //'The values of the log and speed are:'//
     +   '#    ID   Dist_veh  time_veh    Speed'/60('-'))

       tmin = 1./(30.*3600.)
       tmin_diff = 5./3600.
      flag1 = 0
      i = 0
      i1 = 1
       i2 = 1
      do while(flag1.eq.0)
      i = i + 1
      flag2 = 0
      do while(flag2.eq.0)
      if(x12_idv(i1).eq.x12_idv(i2+1))then
      i2= i2+1
      else
      flag2= 2
      end if
      end do
      atr_v_id(i) = x12_idv(i1)
```

```
      if(i1.eq.i2) then
      casev(i) = 1
      atr_t_veh(i) = x12_tv(i1) +
     +          (x_atr - x12_lgv(J4))/(1000*a_sp_veh)* dir
      atr_v_sp(i) = 0.0
       elseif(i2.gt.i1) then
      flag3 = 0
      do k = i1,i2-1
      if((x12_lgv(k).ge.x_atr.and.
     +      x12_lgv(k+1).le.x_atr).or.
     +     (x12_lgv(k).le.x_atr.and.
     +      x12_lgv(k+1).ge.x_atr))then
      spd = a_sp_veh
      atr_t_veh(i) = x12_tv(k) +
     +              (x_atr - x12_lgv(k))/(1000*spd)*dir
      atr_v_sp(i) = x12_spv(k+1)
      flag3 = 1
      casev(i) = 2
      end if
      end do
      if(flag3.eq.0) then
      l_i1 = x12_lgv(i1)
      l_i2 = x12_lgv(i2)
      if (dir.gt.0) then
      if(l_i2.le.x_atr) then
      spd = a_sp_veh
      atr_t_veh(i) = x12_tv(i2) +
     +   (x_atr - x12_lgv(i2))/(1000*spd)*dir
      atr_v_sp(i) = x12_spv(i2)
      casev(i) = 3
      elseif(l_i1.ge.x_atr) then
      spd = a_sp_veh
      atr_t_veh(i) =x12_tv(i1) +
     +   (x_atr - x12_lgv(i1))/(1000*spd)*dir
      atr_v_sp(i) = x12_spv(i1+1)
      casev(i) = 4
      end if
      else if(dir.lt.0)then
      if(l_i1.le.x_atr) then
      spd = a_sp_veh
      atr_t_veh(i) =x12_tv(i1) +
     +   (x_atr - x12_lgv(i1))/(1000*spd)*dir
      atr_v_sp(i) = x12_spv(i1+1)
      casev(i) = 5
      elseif(l_i2.ge.x_atr) then
      spd = a_sp_veh
      atr_t_veh(i) =x12_tv(i2) +
     +             (x_atr - x12_lgv(J5))/(1000*spd)*dir
      atr_v_sp(i) = x12_spv(i2)
      casev(i) = 6
      end if
      end if
      end if
      else
      print*,'
      print*,'**  SOME THING IS WRONG IN THE CHECK   **'
      print*,'**  AT VEHICLE # ',i,'          **'
      print*,'*******************************************'
      end if
```

```
      if(atr_t_veh(i).gt.tmaxv_x1x2) tmaxv_x1x2 = atr_t_veh(i)
      if(atr_t_veh(i).lt.tminv_x1x2) tminv_x1x2 = atr_t_veh(i)
      if(i2.eq.x12_veh) flag1 = 10
      i1 = i2+1
      i2 = i1
      end do
      print*,'
      print*,'** Direction is  = ', dir,'     **'
      print*,'** # of vehicles = ', i , '     **'
      print*,'
      print*, '#   ID   Time    Speed    Case '
      do k = 1,i
      write(6,130)k,atr_v_id(k),atr_t_veh(k),atr_v_sp(k),casev(k)
      end do
      print140,tminv_x1x2, tmaxv_x1x2
140   format(60('-')/'For this segment Tmin is = ',f11.7/
     +  '          &     Tmax is = ',f11.7//
     +   60('-')/'Vehicles that are included in the count are',
     +   /60('-')/' #    ID   SPEED(kmph)   '/60('-'))
130   format(i5, i7,3x,f12.7,f9.3,i4)
      i_sp = 0
      sum_sp = 0
      do k = 1,i
      write(6,130)k,atr_v_id(k),atr_t_veh(k),atr_v_sp(k),casev(k)
      if(atr_v_sp(k).gt.0) then
      i_sp = i_sp + 1
      sum_sp = sum_sp + atr_v_sp(k)
      end if
      end do
      a_sp_in = sum_sp / i_sp
      print160,a_sp_in, i_sp
160   format(60('-')/'The average speed of these vehicles is:',
     +      f9.4,' kmph ',/This is for',i4,' Vehicles'/60('-')/)
150   format(2(i5),f11.4)
      print*,'Cars with Speed = ',a_sp_in ,' Speed=',i_sp
      return
      end
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
     ! NOTES
     ! This Subroutine is based on the fact the same vehilce that
     ! appears more than once is organized in the format that the
     ! the vehicle's first appearance is listed firts.
     ! This subroutine is only for the vehicles that are between
     ! the X1-X2 locations.
     ! This subroutine brings all the vehicles back to the ATR
     ! using the individual speeds of vehicles.
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine bring_to_atr_x1x2(x12_veh,x12_lgv,x12_idv,
     +     x12_tv,x12_spv,
     +     dir,a_sp_veh,x_atr,x1,x2,
     +     tminv_x1x2,tmaxv_x1x2,
     +    atr_v_id,atr_t_veh,atr_v_sp,
     +    i,a_sp_in,i_sp)

      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer dir,x12_veh,k,i,i1,i2,flag1,flag2,flag3,iii,i_sp
      integer x12_idv(x12_veh), atr_v_id(400),casev(400)
      integer id_in_t(400)
```

```
      real sp_in_t(400)
      real x12_lgv(x12_veh),x12_spv(x12_veh),x12_tv(x12_veh)
      real atr_t_veh(x12_veh),atr_v_sp(400)
      real a_sp_veh,spd
      real x_atr,x1,x2,tminv_x1x2,tmaxv_x1x2
      real l_i1, l_i2
      real tmin,tmin_diff,t_diff,t_diff2,tt
      real a_sp_in,sum_sp
      real log_min,log_max
      i_sp = 0
      spd = 0
      tminv_x1x2 = 9999.9999
      tmaxv_x1x2 = 0
      print13,x_atr,x1,x2

   13 format(60('-'),/'ATR dist inside the subroutine is:',f9.1,
     +   /'Start distance for count is = ',f11.5,
     +   /'End  distance for count is  = ',f11.5,
     +   //'The values of the log and speed are:'//
     +   '#    ID   Dist_veh  time_veh    Speed'/60('-'))

      tmin = 1./(30.*3600.)
      tmin_diff = 5./3600.
      flag1 = 0
      i = 0
      i1 = 1
      i2 = 1
      do while(flag1.eq.0)
      i = i + 1
      flag2 = 0
      do while(flag2.eq.0)
      if(x12_idv(i1).eq.x12_idv(i2+1))then
      i2= i2+1
      else
      flag2= 2
      end if
      end do
      atr_v_id(i) = x12_idv(i1)
      if(i1.eq.i2) then
      casev(i) = 1
      atr_t_veh(i) = x12_tv(i1) +
     +         (x_atr - x12_lgv(#REF!))/(1000*a_sp_veh)* dir
      atr_v_sp(i) = 0.0
        elseif(i2.gt.i1) then
      flag3 = 0
      do k = i1,i2-1
      if((x12_lgv(k).ge.x_atr.and.
     +        x12_lgv(k+1).le.x_atr).or.
     +       (x12_lgv(k).le.x_atr.and.
     +        x12_lgv(k+1).ge.x_atr))then
      spd = x12_spv(k+1)
      if(x12_spv(k+1).eq.0) spd = a_sp_veh
      atr_t_veh(i) = x12_tv(k) +
     +       (x_atr - x12_lgv(k))/(1000*spd)*dir
      atr_v_sp(i) = x12_spv(k+1)
      flag3 = 1
      casev(i) = 2
      end if
      end do
```

```
      if(flag3.eq.0) then
      l_i1 = x12_lgv(i1)
      l_i2 = x12_lgv(i2)
      if (dir.gt.0) then
      if(l_i2.le.x_atr) then
      spd = x12_spv(i2)
      if(x12_spv(i2).eq.0) spd = a_sp_veh
      atr_t_veh(i) = x12_tv(i2) +
      +    (x_atr - x12_lgv(i2))/(1000*spd)*dir
      atr_v_sp(i) = x12_spv(i2)
      casev(i) = 3
      elseif(l_i1.ge.x_atr) then
      spd = x12_spv(i1+1)
      if(x12_spv(i1+1).eq.0) spd = a_sp_veh
      atr_t_veh(i) = x12_tv(i1) +
     +   (x_atr - x12_lgv(i1))/(1000*spd)*dir
      atr_v_sp(i) = x12_spv(i1+1)
      casev(i) = 4
      end if
      else if(dir.lt.0)then
      if(l_i1.le.x_atr) then
       spd = x12_spv(i1+1)
      if(x12_spv(i1+1).eq.0) spd = a_sp_veh
       atr_t_veh(i) = x12_tv(i1) +
     +      (x_atr - x12_lgv(i1))/(1000*spd)*dir
      atr_v_sp(i) = x12_spv(i1+1)
      casev(i) = 5
      elseif(l_i2.ge.x_atr) then
       spd = x12_spv(i2)
       if(x12_spv(i2).eq.0) spd = a_sp_veh
       atr_t_veh(i) = x12_tv(i2) +
     +          (x_atr - x12_lgv(J216))/(1000*spd)*dir
      atr_v_sp(i) = x12_spv(i2)
      casev(i) = 6
      end if
      end if
      end if
      else
      print*,'                                    '
      print*,'** SOME THING IS WRONG IN THE CHECK  **'
      print*,'** AT VEHICLE # ',i,'           **'
      print*,
      end if
      if(atr_t_veh(i).gt.tmaxv_x1x2) tmaxv_x1x2 = atr_t_veh(i)
      if(atr_t_veh(i).lt.tminv_x1x2) tminv_x1x2 = atr_t_veh(i)
      if(i2.eq.x12_veh) flag1 = 10
      i1 = i2+1
      i2 = i1
      end do
      print*,
      print*,'** Direction is   = ', dir,'      **'
      print*,'** # of vehicles = ', i , '     **'
      print*,
      print*, '#  ID  Time    Speed    Case '
      do k = 1,i
      write(6,130)k,atr_v_id(k),atr_t_veh(k),atr_v_sp(k),casev(k)
      end do
  130 format(i5, i7,3x,f12.7,f9.3,i4)
```

```
          print140,tminv_x1x2, tmaxv_x1x2
      140 format(60('-')/'For this segment Tmin is = ',f11.7/
     +    '          &    Tmax is = ',f11.7//
     +      60('-')/'Vehicles that are included in the count are',
     +     /60('-')/'  #    ID    SPEED(kmph)   '/60('-'))
          i_sp = 0
          sum_sp = 0
          do k = 1,i
          write(6,130)k,atr_v_id(k),atr_t_veh(k),atr_v_sp(k),casev(k)
          if(atr_v_sp(k).gt.0) then
          i_sp = i_sp + 1
          sum_sp = sum_sp + atr_v_sp(k)
          end if
          end do
          a_sp_in = sum_sp / i_sp
          print160,a_sp_in, i_sp
      160 format(60('-')/'The average speed of these vehicles is:',
     +     f9.4,' kmph ',/'This is for',i4,' Vehicles'/60('-')/)
      150 format(2(i5),f11.4)
          print*,'Cars with Speed = ',a_sp_in ,' Speed=',i_sp
          return
          end
          !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine cnt_type(f_type)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer iof, f_type, ior
      iof = 0
      do while(iof.eq.0)
      print 133
 133  format(/66('=')/
     +  4x,'You have 2 choices to choose from.  These are: '//
     +  4x,'1- Enter (X,Y) of ATR and t1,t2 data. '//
     +  4x,'2- Enter (X,Y) of X1, X2 points and t value.'/
     +  4x,'   This will compute traffic measures as in 1'/
     +  4x,'   but after eliminating the values outside X1, X2'//
     +  4x,'PLEASE Enter which method you want 1,  or 2 : ',$)
      read(5,*,iostat = ior)f_type
      if(ior.eq.0)then
      print*,'**********************************************************'
      if(f_type.ne.1.and.f_type.ne.2)then
      print*,'*******************************************'
      print*,'****   Invalid data try again.      ****'
      print*,'****   Hit return to continue       ****'
      print*,'*******************************************'
      read*
      else
      iof = 9
      end if
      else
      print*,'*******************************************'
      print*,'****   Invalid data try again.      ****'
      print*,'****   Hit return to continue       ****'
      print*,'
      read*
      end if
      end do
      return
      end
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine out_times(t1,t2,tminc,tmaxc,tmint,tmaxt)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      real t1,t2,tminc,tmaxc,tmint,tmaxt
      print 31,  t1,t2,tminc,tmaxc,tmint,tmaxt
  31  format(/'The value of t1 is = ',f9.5,' and t2 is = ',f9.5/
     +      'The times of cars, min = ',f9.5,'  and max =',f9.5/
     +      'The times of trucks, min = ',f9.5,'  and max =',f9.5)
      return
      end
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
       subroutine check_cl_limits(dir,t1,t2,tminc,tmaxc,tmint,tmaxt,
     +               a_sp_cars,a_sp_trks,dis_atr,
     +               cl_1st,cl_last,
     +               tc_st,tc_end,tt_st,tt_end)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer dir
```

```
      real t1,t2,tminc,tmaxc,tmint,tmaxt
      real a_sp_cars,a_sp_trks,dis_atr, cl_1st,cl_last
      real tc_st,tc_end,tt_st,tt_end
      real cl_st_car_sp,cl_end_car_sp,cl_st_trk_sp,cl_end_trk_sp
      cl_st_car_sp =tc_st+(dis_atr-cl_1st)/(1000*a_sp_cars)*dir
      cl_end_car_sp=tc_end+(dis_atr-cl_last)/(1000*a_sp_cars)*dir
      cl_st_trk_sp =tt_st+(dis_atr-cl_1st)/(1000*a_sp_trks)*dir
      cl_end_trk_sp=tt_end+(dis_atr-cl_last)/(1000*a_sp_trks)*dir
      if(tminc.gt.cl_st_car_sp)  tminc =  cl_st_car_sp
      if(tmaxc.lt.cl_end_car_sp) tmaxc =  cl_end_car_sp
      if(tmint.gt.cl_st_trk_sp)  tmint =  cl_st_trk_sp
      if(tmaxt.lt.cl_end_trk_sp) tmaxt =  cl_end_trk_sp
      return
      end
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine save_x12_t12(dis_x1,dis_x2,t1,t2,dx1,dx2,tx1,tx2)
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      real dis_x1,dis_x2,t1,t2,dx1,dx2,tx1,tx2
      tx1 = t1
      tx2 = t2
      dx1 = dis_x1
      dx2 = dis_x2
      return
      end
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine  check_t1t2(t1,t2,tminc,tmaxc,tmint,tmaxt,fail)
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer fail
      real t1,t2,tminc,tmaxc,tmint,tmaxt
      fail = 0
      if(tminc.gt.t1) print 21
      if(tminc.gt.t1) fail = 1
      if(tmaxc.lt.t2) fail = 1
      if(tmaxc.lt.t2) print 22
      if(tmint.gt.t1) print 23
      if(tmint.gt.t1) fail = 1
      if(tmaxt.lt.t2) fail = 1
      if(tmaxt.lt.t2) print 24
   21 format(/'Time t1 is smaller than the limit of the cars. '/)
   22 format(/'Time t2 is greater than the limit of the cars. '/)
   23 format(/'Time t1 is smaller than the limit of the trucks. '/)
   24 format(/'Time t2 is greater than the limit of the trucks. '/)
      return
      end
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine  check_t1t2_x1x2(t1,t2,
     +       dir,a_sp_cars,a_sp_trks,
     +       dis_atr,t0,dx1,dx2,tx1,tx2)
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer dir
      real t1,t2,tx1c_atr,tx2c_atr,tx1t_atr,tx2t_atr
      real a_sp_cars,a_sp_trks,center_1,center_lst
      real dis_atr,t0,dx1,dx2,tx1,tx2
      tx1c_atr = tx1 + (dis_atr - dx1)/(1000.*a_sp_cars)*dir
      tx2c_atr = tx2 + (dis_atr - dx2)/(1000.*a_sp_cars)*dir
      tx1t_atr = tx1 + (dis_atr - dx1)/(1000.*a_sp_trks)*dir
```

```
      tx2t_atr = tx2 + (dis_atr - dx2)/(1000.*a_sp_trks)*dir
      if(tx1t_atr.lt.tx2t_atr)then
      t1 = tx1t_atr
      if(tx1c_atr.gt.tx1t_atr) t1 = tx1c_atr
      t2 = tx2t_atr
      if(tx2c_atr.lt.tx2t_atr) t2 = tx2t_atr
      else
      t1 = tx2t_atr
      if(tx2c_atr.gt.tx2t_atr) t1 = tx2c_atr
      t2 = tx1t_atr
      if(tx1c_atr.lt.tx1t_atr) t2 = tx1t_atr
      end if
      return
      end
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine swap_x1x2(x1,x2,dir)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer  dir
      real  x1,x2,swapx12
      if(x1*dir.gt.x2*dir)then
      swapx12 = x1
      x1 = x2
      x2 = swapx12
      end if
      return
      end
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine check_x1x2(x1,x2,dir,log_cl,numcl)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer numcl, dir
      real log_cl(numcl), x1,x2,swapx12
      if(x1*dir.gt.x2*dir)then
      swapx12 = x1
      x1 = x2
      x2 = swapx12
      end if
      if(dir.gt.0) then
      if(x1*dir.lt.dir*log_cl(2))then
      print 16
   16 format(//,'The first point, X1, is before the limits of'/
     +    'the center line first point.'/)
      STOP
      end if
      if(x2*dir.gt.dir*log_cl(numcl-1))then
      print 17
   17 format(//,'The second point, X2, is before the limits of'/
     +        'the center line last point.'/)
      STOP
      end if
      else if(dir.lt.0)then
      if(x1*dir.lt.dir*log_cl(numcl-1))then
      print 16
      STOP
      end if
      if(x2*dir.gt.dir*log_cl(2))then
      print 17
      STOP
```

```
      end if
      end if
      return
      end
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
       This subroutine will eleminate all the vehicles that are outside
       X1-X2 limits.
       After this then we will bring the vehicles back to the ATR
       locations and find the times.
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine veh_in_x1x2(dir,t1,t2,d_x1,d_x2,veh,n_veh_id,
     +    n_log_veh,n_veh_time_id,veh_sp,
     +    x12_veh,x12_idv,x12_lgv,x12_tv,x12_spv)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer veh,x12_veh,dir
      integer n_veh_id(veh),x12_idv(200)
      real t1,t2, d_x1,d_x2,swapt
      real n_log_veh(veh),n_veh_time_id(veh),veh_sp(veh)
      real x12_lgv(200),x12_tv(200),x12_spv(200)
      integer k
      if(t1.gt.t2)then
      swapt = t1
      t1 = t2
      t2 = swapt
      end if
      do k = 1, veh
      if(k.eq.1.and.n_veh_id(k).eq.n_veh_id(k+1))then
      veh_sp(k) = veh_sp(k+1)
      elseif(k.lt.veh.and.n_veh_id(k).eq.n_veh_id(k+1).
     + and.n_veh_id(k).ne.n_veh_id(k-1))then
      veh_sp(k) = veh_sp(k+1)
      end if
      end do
      x12_veh = 0
      do k = 1,veh
      if(n_veh_time_id(k).ge.t1.and.n_veh_time_id(k).le.t2.and.
     +   n_log_veh(k)*dir.ge.d_x1*dir.and.
     +   n_log_veh(k)*dir.le.d_x2*dir) then
      x12_veh = x12_veh + 1
      x12_idv(x12_veh) =  n_veh_id(k)
      x12_lgv(x12_veh) =  n_log_veh(k)
      x12_tv(x12_veh)  =  n_veh_time_id(k)
      x12_spv(x12_veh) =  veh_sp(k)
      end if
      end do
      return
      end
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine which_sp(speed_type)
      !!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      implicit none
      integer ie, speed_type
  165 print*,'
      print*,'Please indicate which speed you want to use to'
      print*,'bring vehicles to the ATR location: '
      print*,'Enter 1 to use average speed of vehicles'
      print*,'Enter 2 to use individual speeds of vehicles'
      print*
```

```
        read(5,*,iostat= ie)speed_type
        if(ie.ne.0.or.(speed_type.ne.1.and.speed_type.ne.2)) then
        print*,'Your response is invalid.  '
        print*,'Renter response '
        go to 165
        end if
        return
        end
        !!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
         subroutine spdtype(a_sp_veh,spdnew,spdyn)
        !!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
         implicit none
        integer iof, spdyn, ior
        real a_sp_veh,spdnew
        iof = 0
         do while(iof.eq.0)
           print 133,a_sp_veh
133 133      format(/66('=')/
     +   4x,'What average speed do you want to use:'//
     +   4x,'1- The average speed of the vehicles which is:',f7.2,
     +   /4x,'2- An average speed that you define? '/
         4x,'PLEASE Enter 1 or 2 to specify your choice  : ',$)
          read(5,*,iostat = ior)spdyn
         if(ior.eq.0)then
           print*,
          if(spdyn.ne.1.and.spdyn.ne.2)then
                     print*,'
                     print*,'****   Invalid data try again.      *****'
                     print*,'****   Hit return to continue       *****'
                     print*,'
                     read*
          else
                     iof = 9
          end if
         else
                    print*,
                    print*,'*****   Invalid data try again.      *****'
                    print*,'*****   Hit return to continue       *****'
                    print*,
                    read*
         end if
         end do
          iof = 0
         if(spdyn.eq.2) then
134     format( //,4x,'Please enter the speed in Kmph : ',$)
         do while(iof.eq.0)
               print134
               read(5,*,iostat = ior)spdnew
          if(ior.ne.0)then
                 print*,
                 print*,'*****   Invalid data try again.      *****'
                 print*,'*****   Hit return to continue       *****'
                 print*,
                 read*
         else
                 iof = 10
         end if
          end do
              else
```

```
         spdnew = 0
       end if
       return
       end
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine cars
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
implicit none
print*,'--------------------------------------------'
print*,'   CARS '
print*,'--------------------------------------------'
return
end
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine trucks
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
implicit none
print*,'--------------------------------------------'
print*,'   TRUCKS  '
print*,'--------------------------------------------'
return
end
!!!!+!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Appendix B. Pattern Recognition for Stationary and Dynamic Pixels – Statistical Description and Program Listings

1 Notations

In this appendix, we provide the detail description of various components of the statistical pattern recognition procedure discussed in Chapter 3. We will use the notations of Section 3.1. However for completeness, we are reproducing some of these notations here.

- B_{ij}: the gray-level of the pixel of size $1m \times 1m$ in row i and column j in the estimated background image B.
- Y_{ij}: the gray-level in the same pixel of the current image.
- B'_{ij}: transformed value of the brightness matching transform $\phi(B_{ij})$ for the background pixel (i,j), where $\phi(\cdot)$ is *unknown* and must be estimated.
- $R_{ij} = Y_{ij} - B'_{ij}$: differenced current image and the transformed background image. Note that $R_{ij} \in [-255, 255]$.
- p_B: distribution of R for pixels that are given to be stationary.
- p_V: distribution of R for pixels that are given to be dynamic.
- X_{ij}: (unobservable pixel labels, in general),

$$X_{ij} = \begin{cases} 1, & \text{if pixel } (i,j) \text{ in the image } Y \text{ is a stationary pixel,} \\ 0, & \text{otherwise, (dynamic pixel).} \end{cases} \tag{1}$$

- $\pi_{ij} = Prob(X_{ij} = 1)$: the *prior* probability that the pixel (i,j) in the current imnage is a stationary pixel.
- The conditional distributions of the diferenced gray-levels:

$$p(R_{ij}|X_{ij} = 1) = p_B(R_{ij}), \text{ (density of the backg. diff.),} \tag{2}$$
$$p(R_{ij}|X_{ij} = 0) = p_V(R_{ij}), \text{ (density of the veh./backg. diff.).} \tag{3}$$

  It is clear that $p_B(\cdot)$ should be an unimodal distribution centered at zero, but $p_V(\cdot)$ depends on the grey-levels of dynamic pixels in the current image (e.g. vehicle could be both lighter and darker than the background pixels, and shadows may or may not be present).

- We can also write down the joint density of R and X as

$$p(R_{ij}, X_{ij}) = \begin{cases} p_B(R_{ij})\pi_{ij}, & \text{if } X_{ij} = 1, \\ p_V(R_{ij})(1 - \pi_{ij}), & \text{if } X_{ij} = 0. \end{cases} \tag{4}$$

- Then the posterior probability of $X_{ij} = 1$ is given by

$$p(X_{ij} = 1|R_{ij}) = \frac{p_B(R_{ij})\pi(X_{ij} = 1)}{p_B(R_{ij})\pi(X_{ij} = 1) + p_V(R_{ij})\pi(X_{ij} = 0)} \tag{5}$$

$$= \frac{p_B(R_{ij})\pi_{ij}}{p_B(R_{ij})\pi_{ij} + p_V(R_{ij})(1 - \pi_{ij})}. \tag{6}$$

2 Functional Form of Various Components

2.1 The Background Transformation, $\phi(\cdot)$

At present, our program alllows for two types of background transformations:

(1) *A monotonic increasing transformation.* This transformation is defined by

$$\phi(B) = \alpha + e^{\beta} * \left(\sum_{b=0}^{B} \exp[\boldsymbol{S}(b)'\boldsymbol{\theta}] - 1 \right). \tag{7}$$

Where $\boldsymbol{S}(b) = (S_1(b), \ldots, S_p(b))$ and $S_1, \ldots, S_p$ are the natural-spline base function (evaluated at b). Various special cases of this monotonic transformation, that have been used by us are give below:

(i) If $\alpha = 0$, $\boldsymbol{\theta} = \boldsymbol{0}$ and $\beta = 0$, $\phi(\cdot)$ is just the identity transformation, i.e, no transformation is used.

(ii) If $\boldsymbol{\theta} = \boldsymbol{0}$ and $\beta = 0$, $\phi(\cdot)$ is just the one parameter transformation, with just a change in location.

(iii) If $\boldsymbol{\theta} = \boldsymbol{0}$ then $\phi(\cdot)$ is a two parameter, linear transformation with intercept α and slope e^{β}.

(iv) With the shift and a slope term and two knots in the natural splines, one has the five parameter transformation.

(2) *A natural-spline transformation.* With $\boldsymbol{S}(B)$ as defined above, this transformation is simply

$$\phi(B) = \alpha + \boldsymbol{S}(B)'\boldsymbol{\theta}. \tag{8}$$

In this case, $\phi(B)$ does not have to be monotone.

Of course, one could investigate other suitable transformations, as well as quantize the transformed variables differently.

2.2 The Background Difference Distribution, $p_B(\cdot)$

We have limited ourselves to two probability distributions at this point:

(1) *Student's t-distribution.* The difference R, given that $X_{ij} = 1$, is assumed to follow a student's t distribution with median zero (location parameter), scale parameter σ and degrees of freedom *df*. Both σ and *df* need to be estimated.

(2) *Gaussian (normal) distribution.* The difference R, given that $X_{ij} = 1$, is assumed to follow a normal distribution with location zero and standard deviation σ, which is estimated from the data.

Recall that the normal distribution is a limiting case of the t-distribution when *df* goes to infinity. In the image processing literature, the folklore is that the residuals R follow the Laplace distribution. We intend to examine this aspect in the future.

2.3 The Vehicle/Background Differences Distribution, $p_V(\cdot)$

Since, one expects a large variety of images, we have limited to ourselves to maximum entropy distribution on a finite interval (the uniform distribution) and a non-parameteric density function to allow for a large number of shapes.

(1) *Uniform distribution.* The difference $Y_{ij} - B'_{ij}$, when $X_{ij} = 0$, is just assumed to be uniform in the range $[-255, 255]$.

(2) *Smooth density (natural-spline).* The difference $Y_{ij} - B'_{ij}$, when $X_{ij} = 0$, is assumed to be smooth and natural-spline function are used to capture the distribution.

$$p_V(R) = \frac{\exp[\boldsymbol{S}(R')'\boldsymbol{\eta}]}{\sum_{r=-255}^{255} \exp[\boldsymbol{S}(R)'\boldsymbol{\eta}]}. \tag{9}$$

As before, $\boldsymbol{S}(R) = (S_1(R), \ldots, S_p(R))$ is a set of base spline functions evaluated at R. Also, note that the distribution should be continuous, but is quantized on a discrete grid ({-255,-254, ... ,255}), with R' representing the nearest integer value of R.

3 Estimation Procedure

First, note that if the X_{ij}'s (stationary or dynamic) were observables, one could estimate (p_B, ϕ) and p_V by the maximum likelihood method, i.e.,

$$\begin{aligned}
(p_B, \phi) &= \arg\max_{p_B,\phi} \sum_{(i,j):X_{ij}=1} \log[p_B(Y_{ij} - \phi(B_{ij}))] && (10)\\
&= \arg\max_{p_B,\phi} \sum_{i,j} X_{ij} \log[p_B(Y_{ij} - \phi(B_{ij}))], && (11)\\
p_V &= \arg\max_{p_V} \sum_{(i,j):X_{ij}=0} \log[p_V(Y_{ij}) - \phi(B_{ij})] && (12)\\
&= \arg\max_{p_V} \sum_{i,j} (1 - X_{ij}) \log[p_V(Y_{ij} - \phi(B_{ij}))]. && (13)
\end{aligned}$$

However, the X_{ij}'s are not observables, and we are basically interested in finding their posterior distribution. Therefore, instead of ing the log likelihood, we e the expected log likelihood, with respect to the missing data (X_{ij}). Thus we are using the E-M algorithm to estimate the unknown parameters of these densities, in an iterative manner.

$$\begin{aligned}
(p_B, \phi) &= \arg\max_{p_B,\phi} \sum_{i,j} p(X_{ij} = 1) \log[p_B(Y_{ij} - \phi(B_{ij}))], && (14)\\
p_V &= \arg\max_{p_V} \sum_{i,j} p(X_{ij} = 0) \log[p_V(Y_{ij} - \phi(B_{ij}))]. && (15)
\end{aligned}$$

The iterative estimation procedure is as follows:

(1) Initilize $\phi = \phi^{(0)}$ (e.g. $\phi^{(0)}(B) = \alpha + B$ — just a shift in the identify transformation). Let $R_{ij}^{(0)} = Y_{ij} - \phi^{(0)}(B_{ij})$ and initilize $p_B = p_B^{(0)}$ and $p_V = p_V^{(0)}$.

(2) Compute the posterior probability distribution of X_{ij}

$$p^{(0)}(X_{ij}|R_{ij}^{(0)}) = \frac{\pi_{ij})p_B^{(0)}(R_{ij}^{(0)})}{\pi_{ij})p_B^{(0)}(R_{ij}^{(0)}) + (1-\pi_{ij})p_V^{(0)}(R_{ij}^{(0)})}. \tag{16}$$

(3) Update P_B and ϕ. Let

$$(p_B^{(1)}, \phi^{(1)}) = \arg\max_{(p_B,\phi)} \sum_{i,j} p^{(0)}(X_{ij}=1|R_{ij}^{(0)}) \log[p_B(Y_{ij} - \phi(B_{ij}))]. \tag{17}$$

(4) Update p_V. Let

$$p_V^{(1)} = \arg\max_{p_V} \sum_{i,j} p^{(0)}(X_{ij}=0|R_{ij}^{(0)}) \log[p_V(Y_{ij} - \phi^{(1)}(B_{ij}))]. \tag{18}$$

(5) Repeat steps (2) to (4) to get $p_B^{(k)}$, $\phi^{(k)}$ and $p_V^{(k)}$ for $k = 2, 3, \ldots$, until $p^{(k)}(X_{ij}|R_{ij})'s$ converge according to the following criteria.

(6) The convergence of the posterior probabilities $p_{ij}^{(k)} = p^{(k)}(X_{ij}=1|R_{ij})$, is judged by the sum of Kullback-Liebler distance between $p_{ij}^{(k-1)}$ and $p_{ij}^{(k)}$, over all the pixels, i.e.,

$$d(k-1,k) = \sum_{i,j} \left[p_{ij}^{(k)} \log\left(\frac{p_{ij}^{(k)}}{p_{ij}^{(k-1)}}\right) + (1-p_{ij}^{(k)}) \log\left(\frac{1-p_{ij}^{(k)}}{1-p_{ij}^{(k-1)}}\right)\right]. \tag{19}$$

Once the value of $d(k-1,k)$ falls below a certain threshold value, we stop and accept the $p_{ij}^{(k)}$ as the converged posterior probabililities.

4 S+ Code

In this section, we first list the generic Splus functions for various components of the pixel classification procedure, which can be called from within S+ session. The we list the S+ code for the implementation of this procedure on the test and scanned images, as described in Section 3.2. Finally, we also list S+ code for generic image processing, including plotting of images, edge detection, and other filters.

4.1 The S+ Motion Detection Code

```
                                        ##########################################
##
## GJ: 19-APR-98 (25-June-98)
##
## S+ Codes for Variuos Components of Pixel Classification Procedure
##
##
###########################################################################

##### FIRST, FOR THE BACKGROUND TRANSFORMATION #####

## all background transformation are evaluated on the grid 0,1,...,255.

## The natural-spline transformation -- not monotonic increasing.
## is of the form: f(x) = a + ns(x,knots) -- if 2 param, then ns() is linear
get.back.ns.trans.base.mat <- function(back,n=6,knots=NULL,pixels=0:255) {
  ## to get the ns() base-matrix (X matrix)
  ## back: the backg. pixel value
  ## n: the number of parameters in the transformation
  if(n==1)
    ## only intercept
    matrix(1,nrow=length(pixels),ncol=1)
  else if(n==2)
    ## intercept and slope
    cbind(rep(1,length(pixels)),pixels)
  else {
    ## intercept and ns-term
    if(is.null(knots))
      knots <- quantile(back,(1:(n-2))/n)
    cbind(rep(1,length(pixels)),ns(pixels,knots=knots,intercept=F))
  }

}


init.back.ns.trans <- function(res,base.mat)
  ## initial estimate of the background trans. parameters.
  c(mean(res,trim=0.5),rep(0,ncol(base.mat)-1))


get.back.ns.trans <- function(param,base.mat) {
  ## param: the parameters in the n-s transformation (beta)
  ## base.mat: the natural-spline matrix (511 rows)
  pred <- as.vector(base.mat %*% param) + 0:255
  pred[pred<0] <- 0
  pred[pred>255] <- 255
  pred

}


##### A monotonic, increasing, transformation.

## transform background -- monotonic increasing transformation:
get.back.mono.trans <- function(param,base.mat=NULL) {
  ## param: the parameters in the transformation (beta)
  ## base.mat: the natural-spline (ns) matrix with 511 rows.
```

```
 pred <- switch(as.character(length(param)),
                '1' = param[1] + 0:255,
                '2' = param[1] + exp(param[2])*(0:255),
                param[1] + exp(param[2]) *
                as.vector(cumsum(exp(base.mat %*% param[-c(1,2)])) -1)
                )
 pred[pred<0] <- 0
 pred[pred>255] <- 255
 names(pred) <- 0:255
 ##round(pred)
 pred  # not round things
}

## get the natural-spline matrix
get.back.mono.trans.base.mat <- function(back,n=2,pixels=0:255,knots=NULL) {
  if(n>=4) { # have at least on knot in the ns() function
    if(is.null(knots))
      knots <- quantile(back,(1:(n-3))/(n-3+1))
    ns(pixels,knots=knots,intercept=F)
  } else {
    NULL
  }
}

## initial estimate of the background trans. parameters.
init.back.mono.trans <- function(res,n=2)
  c(mean(res,trim=0.5),rep(0,n-1))


## plot the background transformation
plot.back.trans <- function(back.trans,img,back,back.probs,thresh=0.5) {
  ## back.trans: the value of the backg. transf. at 0,1,...,255

  plot(c(0,255),c(0,255),type="n",xlab="Background image (pixel value)",
       ylab="New image (pixel value)")
  ind <- back.probs>thresh
  points(back[ind],img[ind],pch=".",cex=par()$cex*1.5,col=2)
  points(back[!ind],img[!ind],pch=1,col=3,cex=par()$cex*0.8)
  abline(a=0,b=1)
  lines(0:255,back.trans,lwd=3,col=1,lty=3)
  fit <- smooth.spline(back,img,w=back.probs,df=10)
  lines(fit,lwd=4,col=3,lty=2)
  key(x=-30,y=310,transparent=T,
      lines=list(lty=c(3,2),lwd=c(3,3),col=c(3,2)),
      text=list(c("Transf.","s-s (df=10)"))
      )
  key(x=256/2,y=310,transparent=T,
      points=list(pch=c(16,1),cex=par()$cex*c(0.6,0.8),col=c(2,3)),
      text=list(c(paste("P(backg.) > ",thresh,sep=""),
        paste("P(backg.) <= ",thresh,sep="")))
      )

  return(invisible(NULL))
 }


 ####          ############################################################

 ##### FOR DENSITIES #####

 ## get the marginal density of the residuals:
 get.marg.dens <- function(weights,res.ind,un.res=(-255):255) {
   ## res.ind: points to un.res -- discreate residuals are un.res[res.ind]
   n <- length(un.res)
   marg.dens <- tapply(c(weights,rep(0,n)),c(res.ind,1:n),sum)
   names(marg.dens) <- un.res
   marg.dens
 }
```

```
## the background difference density -- t-density:
get.back.t.dens <- function(param,res) {
  dt(res/exp(param[1]),df=exp(param[2]))/exp(param[1])
}


## compute initial estimate for the backgr. difference density
## based on residuals only when using t-density:
init.back.t.dens <- function(res,df=5,red=0.05) {
  ## initial estimates for the t-distribution:
  tmp <- abs(res)
  res <- res[tmp <= quantile(tmp,1-red)]
  c(log(sqrt(var(res)/(df/(df-2)))),
    log(df))
}


## the background difference normal density -- don't need this
get.back.norm.dens <- function(param,res)
  dnorm(res,0,param)


## compute the vehicle density for -255,..., 255
get.veh.dens <- function(param,base.mat,un.res=(-255):255) {
  ## using natural-spline to construct density:
  ## param: beta in X*beta
  ## base.mat: the natural b-spline matrix, X
  probs <- as.vector(exp(base.mat%*%param))
  names(probs) <- un.res
  probs/sum(probs)
}


get.veh.dens.base.mat.and.tot.probs <- function(res.ind,back.probs,
                                                q.probs=(1:5)/6,
                                                un.res=(-255):255) {
  ## returns the ns base matrix, the tot. veh. probs for each pixel value.
  ## q.probs (quantiles) are used to find the knots to use in ns()

  if(!missing(res.ind) && !missing(back.probs))
    tot.veh.probs <- get.marg.dens(1-back.probs,res.ind)
  else
    tot.veh.probs <- get.marg.dens(rep(1,length(un.res)),1:length(un.res))
  cum.tot.veh.probs <- cumsum(tot.veh.probs/sum(tot.veh.probs))
  ## find one quantile:
  one.quantile <- function(prob,x)
    rev(as.numeric(names(x))[x<=prob])[1]
  all.quant <- sapply(q.probs,one.quantile,x=cum.tot.veh.probs)
  ## then, get the base matrix:
  ##base.mat <- bs((-255):255,knots=all.quant,int=F)[,-length(q.probs)-3]
  base.mat <- ns(un.res,knots=all.quant,int=F)
  return(base.mat=base.mat,tot.veh.probs=tot.veh.probs)
}


## plot back. and veh. dens:
plot.back.dens <- function(res,back.probs,back.dens) {
  ## back.dens: the density evaluated at (-255):255

  use.breaks <- seq(-255.5,255.5,by=7)
  res.breaks <- cut(res,breaks=use.breaks)
  bg.hist <- tapply(back.probs,res.breaks,sum)
  bg.hist[is.na(bg.hist)] <- 0
  bg.hist <- bg.hist/(7*sum(bg.hist))
  pix <- (-255):255
  plot(c(-255,255),c(0,max(back.dens,bg.hist)),type="n",
       xlab="Pixel difference (New - Backg.)", ylab="Density")
  panel.histogram(use.breaks,c(NA,bg.hist),border=-1)
  lines(pix,back.dens,lwd=3,col=3)
```

```
}


plot.veh.dens <- function(res,back.probs,veh.dens) {
  ## veh.dens: the veh./backg. diff. density evaluated at (-255):255

  use.breaks <- seq(-255.5,255.5,by=7)
  res.breaks <- cut(res,breaks=use.breaks)
  veh.hist <- tapply(1-back.probs,res.breaks,sum)
  veh.hist[is.na(veh.hist)] <- 0
  veh.hist <- veh.hist/(7*sum(veh.hist))
  pix <- (-255):255
  plot(c(-255,255),c(0,max(veh.dens,veh.hist)),type="n",
       xlab="Pixel difference (New - Backg.)", ylab="Density")
  panel.histogram(use.breaks,c(NA,veh.hist),border=-1)
  lines(pix,veh.dens,lwd=3,col=3)
}


##########################################################################

##### ESTIMATING AND UPDATING #####

## estimate both the background difference denstiy and transformation
## when using the t-density and monotonic backg. transformation:
est.back.t.dens.and.mono.trans <-
  function(img,back,back.probs,param.start,
           back.trans.base.mat) {

  ## the negative log-likelihood:
  opt.func <- function(param) {
    trans.bg <- get.back.mono.trans(param[-c(1,2)],base.mat=back.trans.base.mat)
    - sum( back.probs * log(get.back.t.dens(param=param[1:2],
                                           res=img - trans.bg[back.ind])))
  }

  ## optimize -- minimize the negative log-likelihood:
  assign("img",img,where=0,immediate=T)
  assign("back",back,where=0,immediate=T)
  assign("back.probs",back.probs,where=0,immediate=T)
  assign("back.trans.base.mat",back.trans.base.mat,where=0,immediate=T)
  assign("back.ind",back+1,where=0,immediate=T)
  fit <- nlminb(start=param.start, objective=opt.func,
                control=nlminb.control(eval.max=400,iter.max=200))
  remove(c("img","back","back.probs","back.trans.base.mat"),where=0)
  remove("back.ind",where=0)

  return(fit)
}


## estimate both the background difference denstiy and transformation
## when using the normal density and monotonic backg. transformation:
est.back.norm.dens.and.mono.trans <-
  function(img,back,back.probs,param.start,
           back.trans.base.mat) {

    ## opt. function: minimizing sum of squares (prop. to neg-loglikelihood)
  opt.func <- function(param) {
    trans.bg <- get.back.mono.trans(param,base.mat=back.trans.base.mat)
    sum( back.probs * (img-trans.bg[back.ind])^2)
  }

  ## optimize -- minimize the negative log-likelihood:
  assign("img",img,where=0,immediate=T)
  assign("back",back,where=0,immediate=T)
  assign("back.probs",back.probs,where=0,immediate=T)
  assign("back.trans.base.mat",back.trans.base.mat,where=0,immediate=T)
  assign("back.ind",back+1,where=0,immediate=T)
  fit <- nlminb(start=param.start, objective=opt.func,
```

```
run.EM.t.and.mono <-
  function(img,back,back.prior.probs=NULL,traffic.dens=0.05,
           back.dens.control=list(param=NULL,df=5,trim=0.5),
           back.trans.control=list(param=NULL,nr.trans.param=5,
             knots=NULL),
           veh.dens.control=list(probs=c(0.05,0.2,0.4,0.6,0.9,0.95)),
           max.iter=20, ask.iter=F, conv.crit=1/10,
           update.veh.dens=T
           ) {
    ## img: the image (the pixels in the image)
    ## back: the current estimate of the background pixels
    ## back.prior.probs: the prior prob for pixel being a background pixel.
    ## traffic.dens: a prior estimate of traffic density (used if
    ##               back.prior.probs is NULL).

    ## The other parameters are input parameters to other functions -- see use

  back.ind <- back+1  # index for the background, color 0 is index 1
  res <- img - back   # current difference (residuals)
  n <- length(res)

  ## first, initial estimate of transformation:
  cat("Getting initial estimate of backg. transf. ...\n")
  back.trans.base.mat <- get.back.mono.trans.base.mat(back=back,
                                              n=back.trans.control$nr.trans.param,
                                              knots=back.trans.control$knots)
  if(is.null(back.trans.control$param))
    back.trans.param <- init.back.mono.trans(res,n=back.trans.control$nr.trans.param)
  else
    back.trans.param <- back.trans.control$param
  back.trans <- get.back.mono.trans(back.trans.param,back.trans.base.mat)

  ## new residuals
  res <- img - back.trans[back.ind]

  ## initial backg. difference density -- if param. missing
  cat("Getting initial estimate of backg. diff. density ...\n")
  if(is.null(back.dens.control$param))
    back.dens.param <- init.back.t.dens(res,df=back.dens.control$df,
                                        red=traffic.dens)
  else
    back.dens.param <- back.dens.control$param
  back.dens <- get.back.t.dens(back.dens.param,res)

  ## initial veh./backg. diff. density:
  cat("Getting initial estimate of veh./backg. diff. density ...\n")
  ## it is just uniform
  veh.stuff <- get.veh.dens.base.mat.and.tot.probs(q.probs=veh.dens.control$probs)
  veh.dens.param <- rep(0,ncol(veh.stuff$base.mat))
  un.veh.dens <- get.veh.dens(veh.dens.param,veh.stuff$base.mat)
  res.ind <- round(res)+256
  veh.dens <- un.veh.dens[res.ind]


  ## initial estimate of back.probs -- if missing
  cat("Getting initial posterior estimates of backg. prob's ...\n")
  if(is.null(back.prior.probs))
    back.prior.probs <- 1-traffic.dens
  back.probs <- update.back.probs(back.prior.probs,
                                  back.dens=back.dens,
                                  veh.dens=veh.dens)
  cat("  Have ",round(sum(back.probs)/n*100,4),
      "% are backg. pixels.\n",sep="")


  ## Start EM
  iter <- T
  nr.iter <- 1
  cat("Starting the EM ...\n")
  while(iter) {
```

```
cat("   Iteration",nr.iter,":\n")

## estimate backg. diff. density and backg. transf.:
cat("     Estimating new backg. transf. and density ...\n")
back.trans.and.dens.fit <-
  est.back.t.dens.and.mono.trans(img,back,back.probs,
                                 param.start=c(back.dens.param,
                                   back.trans.param),
                                 back.trans.base.mat=
                                 back.trans.base.mat)
back.trans.param <- back.trans.and.dens.fit$param[-c(1,2)]
cat("       Backg. transf. param. are",
    round(back.trans.param,4),"\n")
back.dens.param <- back.trans.and.dens.fit$param[c(1,2)]
cat("       Backg. diff. density param. are",
    round(exp(back.dens.param),4),"\n")
back.trans <- get.back.mono.trans(back.trans.param,
                             back.trans.base.mat)
res <- img - back.trans[back.ind]  # new residuals

## estimate veh./backg. dens. diff.:
if(update.veh.dens) {
  cat("     Estimating the veh/backg. density ...\n")
  res.ind <- round(res) + 256  # gray-value of -255 has index 1
  veh.stuff <- get.veh.dens.base.mat.and.tot.probs(res.ind,back.probs,
                                                   veh.dens.control$probs)
  veh.dens.fit <- est.veh.dens(veh.stuff$tot.veh.probs,veh.dens.param,
                               veh.stuff$base.mat)
  veh.dens.param <- veh.dens.fit$param
  un.veh.dens <- get.veh.dens(veh.dens.param, veh.stuff$base.mat)
  veh.dens <- un.veh.dens[res.ind]
}

## Update back.probs:
cat("     Update the backg. probabilities ...\n")
new.back.probs <- update.back.probs(back.prior.probs,back.dens,veh.dens)
cat("       Have ",round(sum(new.back.probs)/n*100,4),
    "% are backg. pixels.\n",sep="")

## compute iteration criteria:
back.probs.diff <- sum(log(new.back.probs/back.probs)*new.back.probs,na.rm=T)+
  sum(log((1-new.back.probs)/(1-back.probs))*(1-new.back.probs),na.rm=T)
cat(   "The convergence criteria is",back.probs.diff,"\n")
back.probs <- new.back.probs

if(ask.iter) {
  ask <- T
  while(ask) {
    answer <- menu(c("To do another iteration.","To stop at this point"),
                   title="Shall we continue?")
    if(answer==1 || answer==2)
      ask <- F
    else
      cat("Select 1 or 2 ...\n")
  }
  if(answer==2)
    iter <- F
} else {
  if(nr.iter >= max.iter || back.probs.diff <= conv.crit)
    iter <- F
}

nr.iter <- nr.iter + 1

}

return(back.probs=back.probs,
       back.dens.param=back.dens.param,
       back.trans.param=back.trans.param,
       back.trans.base.mat=back.trans.base.mat,
```

```
        veh.dens.param=veh.dens.param,
        veh.dens.base.mat=veh.stuff$base.mat)

}


###########################################################################


## Estimate everything: transformation, densities and weights
## using the EM algorithm.
## This is for the case when:
##   (1) The backg. diff. density is a normal density
##   (2) The backg. transformation is monotonic increasing
##   (3) The veh./backg. diff. density can either by estimated or unif.

## VERY slow function:

run.EM.norm.and.mono <-
  function(img,back,back.prior.probs=NULL,traffic.dens=0.05,
           back.dens.control=list(param=NULL),
           back.trans.control=list(param=NULL,nr.trans.param=5,
             knots=NULL),
           veh.dens.control=list(probs=c(0.05,0.2,0.4,0.6,0.9,0.95)),
           max.iter=20, ask.iter=F, conv.crit=1/100,
           update.veh.dens=T
           ) {
    ## img: the image (the pixels in the image)
    ## back: the current estimate of the background pixels
    ## back.probs: the prior prob for pixel being a background pixel.
    ## traffic.dens: a prior estimate of traffic density (prop. of pixels
    ## belonging to vehicles in the new image.

    ## The other parameters are input parameters to other functions -- see use

  back.ind <- back+1  # index for the background, color 0 is index 1
  res <- img - back   # current difference (residuals)
  n <- length(res)

  ## first, initial estimate of transformation:
  cat("Getting initial estimate of backg. transf. ...\n")
  back.trans.base.mat <- get.back.mono.trans.base.mat(back=back,
                                          n=back.trans.control$nr.trans.param,
                                          knots=back.trans.control$knots)
  if(is.null(back.trans.control$param))
    back.trans.param <- init.back.mono.trans(res,n=back.trans.control$nr.trans.param)
  else
    back.trans.param <- back.trans.control$param
  back.trans <- get.back.mono.trans(back.trans.param,back.trans.base.mat)

  ## new residuals
  res <- img - back.trans[back.ind]

  ## initial backg. diff density (the SD in the normal)
  cat("Getting initial estimate of the SD in the backg. diff. density ... \n")
  tmp <- abs(res)
  tmp <- res[tmp <= quantile(tmp,1-traffic.dens)]
  back.dens.param <- sqrt(sum(tmp^2)/length(tmp))
  back.dens <- dnorm(res,0,back.dens.param)

  ## initial veh./backg. diff. density:
  cat("Getting initial estimate of veh./backg. diff. density (unif.) ...\n")
  ## it is just uniform
  veh.stuff <- get.veh.dens.base.mat.and.tot.probs(q.probs=veh.dens.control$probs)
  veh.dens.param <- rep(0,ncol(veh.stuff$base.mat))
  un.veh.dens <- get.veh.dens(veh.dens.param,veh.stuff$base.mat)
  res.ind <- round(res)+256
  veh.dens <- un.veh.dens[res.ind]
```

```
## initial estimate of back.probs -- if missing
cat("Getting initial estimates of backg. prob's ...\n")
if(is.null(back.probs))
  back.prior.probs <- 1-traffic.dens
back.probs <- update.back.probs(back.prior.probs,
                               back.dens=back.dens,
                               veh.dens=veh.dens)
cat("  Have ",round(sum(back.probs)/n*100,4),
    "% are backg. pixels.\n",sep="")


## Start EM
iter <- T
nr.iter <- 1
cat("Starting the EM ...\n")
while(iter) {
  cat("   Iteration",nr.iter,":\n")

  ## estimate backg. diff. density and backg. transf.:
  cat("     Estimating new backg. transf. and density ...\n")
  back.trans.and.dens.fit <-
    est.back.norm.dens.and.mono.trans(img,back,back.probs,
                                      param.start=back.trans.param,
                                      back.trans.base.mat=back.trans.base.mat)
  back.trans.param <- back.trans.and.dens.fit$param
  cat("       Backg. transf. param. are",
      round(back.trans.param,4),"\n")
  back.trans <- get.back.mono.trans(back.trans.param,
                                    back.trans.base.mat)
  res <- img - back.trans[back.ind]  # new residuals
  back.dens.param <- sqrt(sum(back.probs*res^2)/sum(back.probs))
  cat("       Backg. diff. density SD is",round(back.dens.param,2),"\n")
  back.dens <- dnorm(res,0,back.dens.param)

  ## estimate veh./backg. dens. diff.:
  if(update.veh.dens) {
    cat("     Estimating the veh/backg. density ...\n")
    res.ind <- round(res) + 256  # gray-value of -255 has index 1
    veh.stuff <- get.veh.dens.base.mat.and.tot.probs(res.ind,back.probs,
                                                     veh.dens.control$probs)
    veh.dens.fit <- est.veh.dens(veh.stuff$tot.veh.probs,veh.dens.param,
                                 veh.stuff$base.mat)
    veh.dens.param <- veh.dens.fit$param
    un.veh.dens <- get.veh.dens(veh.dens.param, veh.stuff$base.mat)
    veh.dens <- un.veh.dens[res.ind]
  }

  ## Update back.probs:
  cat("     Update the backg. probabilities ...\n")
  new.back.probs <- update.back.probs(back.prior.probs,back.dens,veh.dens)
  cat("       Have ",round(sum(new.back.probs)/n*100,4),
      "% are backg. pixels.\n",sep="")

  ## compute iteration criteria:
  back.probs.diff <-
  sum(log(new.back.probs/back.probs)*new.back.probs,na.rm=T)+
    sum(log((1-new.back.probs)/(1-back.probs))*(1-new.back.probs),na.rm=T)
  cat(   "The convergence criteria is",back.probs.diff,"\n")
  back.probs <- new.back.probs

  if(ask.iter) {
    ask <- T
    while(ask) {
      answer <- menu(c("To do another iteration.","To stop at this point"),
                     title="Shall we continue?")
      if(answer==1 || answer==2)
        ask <- F
      else
        cat("Select 1 or 2 ...\n")
    }
```

```
      if(answer==2)
        iter <- F
    } else {
      if(nr.iter >= max.iter || back.probs.diff <= conv.crit)
        iter <- F
    }

    nr.iter <- nr.iter + 1

  }

  return(back.probs=back.probs,
         back.dens.param=back.dens.param,
         back.trans.param=back.trans.param,
         back.trans.base.mat=back.trans.base.mat,
         veh.dens.param=veh.dens.param,
         veh.dens.base.mat=veh.stuff$base.mat)

}


####            ##########################################       #######

##### USING NORMALD BACKG. DENSITY AND N-S TRANSFORMATION #####

## Estimate everything: transformation, densities and weights
## using the EM algorithm.
## This is for the case when:
##   (1) The backg. diff. density is a normal density
##   (2) The backg. transformation is natural-spline
##   (3) The veh./backg. diff. density can either by estimated or unif.

## This is the fastest function:

run.EM.norm.and.ns <-
  function(img,back,back.prior.probs=NULL,traffic.dens=0.05,
           back.dens.control=list(param=NULL),
           back.trans.control=list(param=NULL,nr.trans.param=5,
             knots=NULL),
           veh.dens.control=list(probs=c(0.05,0.2,0.4,0.6,0.8,0.95)),
           max.iter=20, ask.iter=F, conv.crit=1/10,
           update.veh.dens=T
           ) {
    ## img: the image (the pixels in the image)
    ## back: the current estimate of the background pixels
    ## back.prior.probs: the prior prob for pixel being a background pixel.
    ## traffic.dens: a prior estimate of traffic density (used if
    ##               back.prior.probs is NULL -- missing).

    ## The other parameters are input parameters to other functions -- see use

    back.ind <- as.vector(back+1) #index for the background, color 0 is index 1
    res.null <- as.vector(img - back)   # the raw difference.
    n <- length(res.null)

    ## first, initial estimate of transformation:
    cat("Getting initial estimate of backg. transf. ...\n")
    back.trans.base.mat <-
      get.back.ns.trans.base.mat(back=back,
                                 n=back.trans.control$nr.trans.param,
                                 knots=back.trans.control$knots)
    if(is.null(back.trans.control$param))
      back.trans.param <- init.back.ns.trans(res.null,back.trans.base.mat)
    else
      back.trans.param <- back.trans.control$param
    back.trans <- get.back.ns.trans(back.trans.param,back.trans.base.mat)
    ## create the X matrix for the lsfit() function
    back.trans.fit.mat <- back.trans.base.mat[back.ind,]

    ## new residuals
```

```
res <- img - back.trans[back.ind]

## initial backg. diff density (the SD in the normal)
cat("Getting initial estimate of the SD in the
backg. diff. density ... \n" )
tmp <- abs(res)
tmp <- res[tmp <= quantile(tmp,1-traffic.dens)]
back.dens.param <- sqrt(sum(tmp^2)/length(tmp))
back.dens <- dnorm(res,0,back.dens.param)

## initial veh./backg. diff. density:
cat("Getting initial estimate of veh./backg. diff. density (unif.) ...\n")
## it is just uniform
veh.stuff <-
get.veh.dens.base.mat.and.tot.probs(q.probs=veh.dens.control$probs)
veh.dens.param <- rep(0,ncol(veh.stuff$base.mat))
un.veh.dens <- get.veh.dens(veh.dens.param,veh.stuff$base.mat)
res.ind <- round(res)+256
veh.dens <- un.veh.dens[res.ind]


## initial estimate of posterior back.probs:
cat("Getting initial estimates of backg. prob's ...\n")
## get the backg. density for residuals:
if(is.null(back.prior.probs))
  back.prior.probs <- 1-traffic.dens  ## can be 1 number
back.probs <- update.back.probs(back.prior.probs,
                                back.dens=back.dens,
                                veh.dens=veh.dens)
cat("  Have ",round(sum(back.probs)/n*100,4),
    "% are backg. pixels.\n",sep="")


## Start EM
iter <- T
nr.iter <- 1
cat("Starting the EM ...\n")
while(iter) {
  cat("   Iteration",nr.iter,":\n")

  ## estimate backg. diff. density and backg. transf.:
  cat("     Estimating new backg. transf. and density ...\n")
  back.trans.and.dens.fit <-
  lsfit(x=back.trans.fit.mat,y=res.null,int=F,wt=back.probs)[c("coef","res")]
  back.trans.param <- back.trans.and.dens.fit$coef
  res <- back.trans.and.dens.fit$res
  cat("       Backg. transf. param. are",
      round(back.trans.param,4),"\n")
  back.dens.param <- sqrt(sum(back.probs*res^2)/sum(back.probs))
  cat("       Backg. diff. density SD is",round(back.dens.param,2),"\n")
  back.dens <- dnorm(res,0,back.dens.param)

  ## estimate veh./backg. dens. diff.:
  if(update.veh.dens) {
    cat("     Estimating the veh/backg. density ...\n")
    res.ind <- round(res) + 256  # gray-value of -255 has index 1
    veh.stuff <- get.veh.dens.base.mat.and.tot.probs(res.ind,back.probs,
                                                     veh.dens.control$probs)
    veh.dens.fit <- est.veh.dens(veh.stuff$tot.veh.probs,veh.dens.param,
                                 veh.stuff$base.mat)
    veh.dens.param <- veh.dens.fit$param
    un.veh.dens <- get.veh.dens(veh.dens.param, veh.stuff$base.mat)
    veh.dens <- un.veh.dens[res.ind]
  }

  ## Update back.probs:
  cat("     Update the backg. probabilities ...\n")
  new.back.probs <- update.back.probs(back.prior.probs,back.dens,veh.dens)
  cat("       Have ",round(sum(new.back.probs)/n*100,4),
      "% are backg. pixels.\n",sep="")
```

```
    ## compute iteration criteria:
    back.probs.diff <-
    sum(log(new.back.probs/back.probs)*new.back.probs,na.rm=T)+
      sum(log((1-new.back.probs)/(1-back.probs))*(1-new.back.probs),na.rm=T)
    cat(   "The convergence criteria is",back.probs.diff,"\n")
    back.probs <- new.back.probs

    if(ask.iter) {
      ask <- T
      while(ask) {
        answer <- menu(c("To do another iteration.","To stop at this point"),
                       title="Shall we continue?")
        if(answer==1 || answer==2)
          ask <- F
        else
          cat("Select 1 or 2 ...\n")
      }
      if(answer==2)
        iter <- F
    } else {
      if(nr.iter >= max.iter || back.probs.diff <= conv.crit)
        iter <- F
    }

    nr.iter <- nr.iter + 1

  }

  return(back.probs=back.probs,
         back.dens.param=back.dens.param,
         back.trans.param=back.trans.param,
         back.trans.base.mat=back.trans.base.mat,
         veh.dens.param=veh.dens.param,
         veh.dens.base.mat=veh.stuff$base.mat)

}
```

4.2 The S+ Code for Implementation on Test and Scanned Images

```
               ##########################################################
##
## Gardar Johannesson: 23-June-98
## file: detecting_motion_comands.s
##
## S comands using the functions in the file detecting_motion.s
## and plotting figures for the file detecting_motion.tex
##
                        ##
## Revised June 1999- By Parag Goel
## attaching sample images to use:
attach("/home/pxg/SATELLITE/Image_analysis/Sample_images/.Data")
## attaching functions to plot images:
attach("/home/pxg/SATELLITE/Image_analysis/.Data")

##                      ####################       ####             ###

##### FIRST, PLOT THE SAMPLE IMAGES #####

## plot I70b photo nr. 56
tmp <- i70b.56
tmp[!i70b.56.cut.ind] <- NA
image.device("postscript",file="detecting_motion_img_56.ps",
             height=8,data.dim=dim(tmp))
plot.image(tmp)
dev.off()
## plot only vehicles:
```

```
tmp[!i70b.56.veh.ind] <- NA
image.device("postscript",file="detecting_motion_img_56_veh.ps",
             height=8,data.dim=dim(tmp))
plot.image(tmp)
dev.off()
## plot the background -- estimated
tmp <- i70b.56.bg
tmp[!i70b.56.cut.ind] <- NA
image.device("postscript",file="detecting_motion_img_56_bg.ps",
             height=8,data.dim=dim(tmp))
plot.image(tmp)
dev.off()

## plot I70b photo nr. 57 -- which was resampled wrt nr.56
tmp <- i70b.57.bi
tmp[!i70b.56.cut.ind] <- NA
image.device("postscript",file="detecting_motion_img_57.ps",
             height=8,data.dim=dim(tmp))
plot.image(tmp)
dev.off()
## plot only vehicles:
tmp[!i70b.57.veh.ind] <- NA
image.device("postscript",file="detecting_motion_img_57_veh.ps",
             height=8,data.dim=dim(tmp))
plot.image(tmp)
dev.off()
## plot the background -- estimated
tmp <- i70b.57.bg.bi
tmp[!i70b.56.cut.ind] <- NA
image.device("postscript",file="detecting_motion_img_57_bg.ps",
             height=8,data.dim=dim(tmp))
plot.image(tmp)
dev.off()

## plot histogram of pixelvalues in sample images:
## for Image A:
tmp <- i70b.56[i70b.56.cut.ind]
trellis.device(postscript,file="detecting_motion_hist_A.ps",
               width=8,height=5,horizontal=F)
histogram(~tmp,breaks=seq(-0.5,255.5,by=2),
          xlab="Pixel reflective value",
          ylab="Density")
dev.off()
## for Image B:
tmp <- i70b.57.bi[i70b.56.cut.ind]
trellis.device(postscript,file="detecting_motion_hist_B.ps",
               width=8,height=5,horizontal=F)
histogram(~tmp,breaks=seq(-0.5,255.5,by=2),
          xlab="Pixel reflective value",
          ylab="Density")
dev.off()
## for Image A, vehicles only:
tmp <- i70b.56[i70b.56.cut.ind & i70b.56.veh.ind]
trellis.device(postscript,file="detecting_motion_hist_A_veh.ps",
               width=8,height=5,horizontal=F)
histogram(~tmp,breaks=seq(-0.5,255.5,by=2),
          xlab="Pixel reflective value",
          ylab="Density")
dev.off()
## for Image B, vehicles only:
tmp <- i70b.57.bi[i70b.56.cut.ind & i70b.57.veh.ind]
trellis.device(postscript,file="detecting_motion_hist_B_veh.ps",
               width=8,height=5,horizontal=F)
histogram(~tmp,breaks=seq(-0.5,255.5,by=2),
          xlab="Pixel reflective value",
          ylab="Density")
dev.off()


##### ESTIMATING THE BACKGROND DIFFERENCE DISTRIBUTION
```

```
##### AND THE TRANSFORMATION

##### First, show the difference in the images as 'images'
tmp <- i70b.56
tmp[!i70b.56.cut.ind] <- NA
tmp <- abs(tmp - i70b.57.bg.bi)
image.device("postscript",file="detecting_motion_img_56m57.ps",
             height=8,data.dim=dim(tmp))
plot.image(255-tmp)
dev.off()
tmp <- i70b.57.bi
tmp[!i70b.56.cut.ind] <- NA
tmp <- abs(tmp - i70b.56.bg)
image.device("postscript",file="detecting_motion_img_57m56.ps",
             height=8,data.dim=dim(tmp))
plot.image(255-tmp)
dev.off()


## Using image A as new image and B as background:
tmp.ind <- i70b.56.cut.ind
tmp.img <- i70b.56[tmp.ind]
tmp.bg <- i70b.57.bg.bi[tmp.ind]
tmp.back.probs <- rep(1,length(tmp.img))
tmp.back.probs[i70b.56.veh.ind[tmp.ind]] <- 0  # 0 weights to vehicle
tmp.res <- tmp.img - tmp.bg

## Use the t-distribution and monotonic transformation:
## only using intercept in the transformation:
tmp.base.mat <- NULL
tmp.start <- c(init.back.t.dens(tmp.res),init.back.mono.trans(tmp.res,n=1))
fit.t.mono.int <- est.back.t.dens.and.mono.trans(tmp.img,tmp.bg,tmp.back.probs,tmp.start,tmp.base.mat)
## and plot:
## (1) the histogram:
tmp <- tmp.img - get.back.mono.trans(fit.t.mono.int$param[-c(1,2)],
                                       tmp.base.mat)[tmp.bg+1]
tmp <- tmp[tmp.back.probs==1]  # where we have background
trellis.device(postscript,file="detecting_motion_hist_t_mono_1.ps",
               width=8,height=5,horizontal=F)
histogram(~tmp,breaks=seq(-255.5,255.5,by=7),
          xlab="Pixel reflective value",
          ylab="Density (%)",
          panel=function(x,y,...) {
            panel.histogram(x,y,border=1,...)
            pix <- (-255):255
            param <- fit.t.mono.int$param
            tmp <- dt(pix/exp(param[1]),exp(param[2]))/exp(param[1])
            lines(pix,7*100*tmp,lwd=3,col=3)
          }
          )
dev.off()
## (2) the transformation:
tmp <- get.back.mono.trans(fit.t.mono.int$param[-c(1,2)],tmp.base.mat)
trellis.device(postscript,file="detecting_motion_trans_t_mono_1.ps",
               width=8,height=8,horizontal=F)
plot.back.trans(tmp,tmp.img,tmp.bg,tmp.back.probs)
dev.off()

## using 5 parameters in the transformation:
tmp.base.mat <- get.back.mono.trans.base.mat(tmp.bg,n=5)
tmp.start <- c(init.back.t.dens(tmp.res),init.back.mono.trans(tmp.res,n=5))
fit.t.mono.ns <- est.back.t.dens.and.mono.trans(tmp.img,tmp.bg,tmp.back.probs,tmp.start,tmp.base.mat)
## and plot:
## (1) the histogram:
tmp <- tmp.img - get.back.mono.trans(fit.t.mono.ns$param[-c(1,2)],
                                       tmp.base.mat)[tmp.bg+1]
tmp <- tmp[tmp.back.probs==1]  # where we have background
trellis.device(postscript,file="detecting_motion_hist_t_mono_2.ps",
               width=8,height=5,horizontal=F)
histogram(~tmp,breaks=seq(-255.5,255.5,by=7),
```

```
          xlab="Pixel reflective value",
          ylab="Density (%)",
          panel=function(x,y,...) {
            panel.histogram(x,y,border=1,...)
            pix <- (-255):255
            param <- fit.t.mono.ns$param
            tmp <- dt(pix/exp(param[1]),exp(param[2]))/exp(param[1])
            lines(pix,7*100*tmp,lwd=3,col=3)
          }
          )
dev.off()
## (2) the transformation:
tmp <- get.back.mono.trans(fit.t.mono.ns$param[-c(1,2)],tmp.base.mat)
trellis.device(postscript,file="detecting_motion_trans_t_mono_2.ps",
               width=8,height=8,horizontal=F)
plot.back.trans(tmp,tmp.img,tmp.bg,tmp.back.probs)
dev.off()

## Use the normal distribution and ns() transformation
## only using intercept in the transformation:
tmp.base.mat <- get.back.ns.trans.base.mat(tmp.bg,n=1)
fit.t.mono.int <- est.back.t.dens.and.mono.trans(tmp.img,tmp.bg,tmp.back.probs,tmp.start,tmp.base.mat)
## and plot:
## (1) the histogram:
tmp <- tmp.img - get.back.mono.trans(fit.t.mono.int$param[-c(1,2)],
                                            tmp.base.mat)[tmp.bg+1]
tmp <- tmp[tmp.back.probs==1]  # where we have background
trellis.device(postscript,file="detecting_motion_hist_t_mono_1.ps",
               width=8,height=5,horizontal=F)
histogram(~tmp,breaks=seq(-255.5,255.5,by=7),
          xlab="Pixel reflective value",
          ylab="Density (%)",
          panel=function(x,y,...) {
            panel.histogram(x,y,border=1,...)
            pix <- (-255):255
            param <- fit.t.mono.int$param
            tmp <- dt(pix/exp(param[1]),exp(param[2]))/exp(param[1])
            lines(pix,7*100*tmp,lwd=3,col=3)
          }
          )
dev.off()
## (2) the transformation:
tmp <- get.back.mono.trans(fit.t.mono.int$param[-c(1,2)],tmp.base.mat)
trellis.device(postscript,file="detecting_motion_trans_t_mono_1.ps",
               width=8,height=8,horizontal=F)
plot.back.trans(tmp,tmp.img,tmp.bg,tmp.back.probs)
dev.off()


###
## create table for the sigma and df of background distribution:
tmp.tab <- data.frame('Scale'=exp(c(fit.t.mono.int$param[1],\\
           fit.t.mono.ns$param[1])),'Df'=exp(c(fit.t.mono.int$param[1],\\
           fit.t.mono.ns$param[1])))
tmp <- tmp.img - cbind(get.back.mono.trans(fit.t.mono.int$\\
       param[-c(1,2)],tmp.base.mat), get.back.mono.trans(\\
       fit.t.mono.ns$param[-c(1,2)],tmp.base.mat))[tmp.bg+1,]
tmp.tab$'SD' <- sqrt(apply(tmp[tmp.back.probs==1,],2,var))
tmp.tab$'25% quantile' <- apply(tmp[tmp.back.probs==1,],2,quantile,probs=0.25)
tmp.tab$'75% quantile' <- apply(tmp[tmp.back.probs==1,],2,quantile,probs=0.75)
tmp.tab$'IQR' <- tmp.tab$'75% quantile' - tmp.tab$'25% quantile'
dimnames(tmp.tab)[[1]] <- c('1 param.','5 param.')
round(tmp.tab,3)

#####
## create table of log-likelihoods and test for better transformation:
tmp.tab <- cbind('nr. of param.'=c(1,5),
                 'log-likelihood'=-c(fit.t.mono.int$obj,fit.t.mono.ns$obj))
tmp.tab <- as.data.frame(tmp.tab)
dimnames(tmp.tab)[[1]] <- c('1 param.','5 param.')
```

```
tmp.tab$'log-lik. diff' <- c(NA,tmp.tab[2,'log-likelihood']-
                              tmp.tab[1,'log-likelihood'])
tmp.tab$'p-value' <- round(c(NA,1-pchisq(tmp.tab$'log-lik. diff'[2],4)))
tmp.tab


## Use the normal distribution and ns() transformation
## only using intercept in the transformation:
tmp.base.mat <- get.back.ns.trans.base.mat(tmp.bg,n=1)
tmp.fit.mat <- tmp.base.mat[tmp.bg+1,]
fit.norm.ns.1p <- lsfit(x=tmp.fit.mat,y=tmp.res,int=F,
                        wt=tmp.back.probs)[c("coef","res")]
## and plot:
## (1) the histogram:
tmp <- tmp.img - get.back.ns.trans(fit.norm.ns.1p$coef,
                                   tmp.base.mat)[tmp.bg+1]
tmp <- tmp[tmp.back.probs==1]  # where we have background
trellis.device(postscript,file="detecting_motion_hist_norm_ns_1.ps",
               width=8,height=5,horizontal=F)
histogram(~tmp,breaks=seq(-255.5,255.5,by=7),
          xlab="Pixel reflective value",
          ylab="Density (%)",
          panel=function(x,y,...) {
            panel.histogram(x,y,border=1,...)
            pix <- (-255):255
            param <- sqrt(sum(tmp.back.probs*fit.norm.ns.1p$res^2) /
                          sum(tmp.back.probs))
            tmp <- dnorm(pix,0,param)
            lines(pix,7*100*tmp,lwd=3,col=3)
          }
          )
dev.off()
## (2) the transformation:
tmp <- get.back.ns.trans(fit.norm.ns.1p$coef,tmp.base.mat)
trellis.device(postscript,file="detecting_motion_trans_norm_ns_1.ps",
               width=8,height=8,horizontal=F)
plot.back.trans(tmp,tmp.img,tmp.bg,tmp.back.probs)
dev.off()

## use 5 parameters in the ns transformation:
tmp.base.mat <- get.back.ns.trans.base.mat(tmp.bg,n=5)
tmp.fit.mat <- tmp.base.mat[tmp.bg+1,]
fit.norm.ns.5p <- lsfit(x=tmp.fit.mat,y=tmp.res,int=F,
                        wt=tmp.back.probs)[c("coef","res")]
## and plot:
## (1) the histogram:
tmp <- tmp.img - get.back.ns.trans(fit.norm.ns.5p$coef,
                                   tmp.base.mat)[tmp.bg+1]
tmp <- tmp[tmp.back.probs==1]  # where we have background
trellis.device(postscript,file="detecting_motion_hist_norm_ns_2.ps",
               width=8,height=5,horizontal=F)
histogram(~tmp,breaks=seq(-255.5,255.5,by=7),
          xlab="Pixel reflective value",
          ylab="Density (%)",
          panel=function(x,y,...) {
            panel.histogram(x,y,border=1,...)
            pix <- (-255):255
            param <- sqrt(sum(tmp.back.probs*fit.norm.ns.5p$res^2) /
                          sum(tmp.back.probs))
            tmp <- dnorm(pix,0,param)
            lines(pix,7*100*tmp,lwd=3,col=3)
          }
          )
dev.off()
## (2) the transformation:
tmp <- get.back.ns.trans(fit.norm.ns.5p$coef,tmp.base.mat)
trellis.device(postscript,file="detecting_motion_trans_norm_ns_2.ps",
               width=8,height=8,horizontal=F)
plot.back.trans(tmp,tmp.img,tmp.bg,tmp.back.probs)
dev.off()
```

```
###########################################################################

### ESTIMATING THE VEHICLE MINUS BACKGROUND DISTRIBUTION


## Using image A as new image and B as background:
tmp.ind <- i70b.56.cut.ind
tmp.img <- i70b.56[tmp.ind]
tmp.bg <- i70b.57.bg.bi[tmp.ind]
tmp.back.probs <- rep(1,length(tmp.img))
tmp.back.probs[i70b.56.veh.ind[tmp.ind]] <- 0  # 0 weights to vehicles
## use the ns transformation with 5 parameters




                          ###################       ####################

##### TESTING ITERATIVE EM PROCEDURE #####

###
## Use a test images:
tmp.true.bg <- matrix(150+20*rnorm(30*20),30,20)  # the true background
tmp.true.bg[,4:7] <- tmp.true.bg[,4:7] - 40
tmp.true.bg[,12:16] <- tmp.true.bg[,12:16] - 30
tmp.true.bg[,19:20] <- tmp.true.bg[,19:20] - 70
tmp.true.bg[tmp.true.bg<0] <- 0
tmp.true.bg[tmp.true.bg>255] <- 255
## the brighness change:
tmp.fit <- smooth.spline(x=c(0,50,100,150,200,255),
                         y=c(0,40,85,120,155,180),df=5)
tmp.trans <- predict(tmp.fit,x=0:255)
## the new image:
tmp.img <- matrix(approx(tmp.trans$x,tmp.trans$y,xout=tmp.true.bg)$y +
                  7*rnorm(30*20), 30,20)
tmp.img[6:12,4:8] <- 5*rnorm(7*5) # moving object nr. 1 (shadow)
tmp.img[6:11,4:7] <- 40+5*rnorm(6*4) # the object nr.1
tmp.img[18:25,12:17] <- 5*rnorm(8*6) # moving object nr. 2 (shadow)
tmp.img[18:24,12:16] <- 170+5*rnorm(7*5) # the object nr 2
tmp.img[tmp.img<0] <- 0
tmp.img[tmp.img>255] <- 255
## number of vehicle pixels
(7*5+8*6)/(30*20)  # approx 14% or 83 pixels
## the observed background
tmp.bg <- tmp.true.bg + 7*rnorm(30*20)

#plot test images
image.device("postscript",file="detecting_motion_test_img.ps",
             height=6,data.dim=dim(tmp))
plot.image(tmp.img)
dev.off()
image.device("postscript",file="detecting_motion_test_bg.ps",
             height=6,data.dim=dim(tmp))
plot.image(tmp.bg)
dev.off()


#####
## use unif. veh. dens. with the same prior (traffic.dens), but different
## transformation
## Use 1 param:
tmp.fit.1 <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                                traffic.dens=0.14,
                                back.trans.control=list(nr.trans.param=1))
```

```
## plot weights:
tmp <- tmp.img
##tmp[] <- round(255*tmp.fit.1$back.probs)
tmp[] <- ifelse(tmp.fit.1$back.probs>=0.5,255,0)
image.device('postscript',file="detecting_motion_test_pp_1.ps",
             data.dim=dim(tmp),height=6)
plot.image(tmp)
dev.off()
## plot background transformation
tmp.back.trans <- get.back.ns.trans(tmp.fit.1$back.trans.param,
                                    tmp.fit.1$back.trans.base.mat)
trellis.device(postscript,file="detecting_motion_test_bt_1.ps",
               width=6,height=6,horizontal=F)
plot.back.trans(tmp.back.trans,tmp.img,tmp.bg,tmp.fit.1$back.probs)
dev.off()

## Use 5 param:
tmp.fit.2 <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                                traffic.dens=0.14,
                                back.trans.control=list(nr.trans.param=5))
## plot weights:
tmp <- tmp.img
##tmp[] <- round(255*tmp.fit.2$back.probs)
tmp[] <- ifelse(tmp.fit.2$back.probs>=0.5,255,0)
image.device('postscript',file="detecting_motion_test_pp_2.ps",
             data.dim=dim(tmp),height=6)
plot.image(tmp)
dev.off()
## plot background transformation
tmp.back.trans <- get.back.ns.trans(tmp.fit.2$back.trans.param,
                                    tmp.fit.2$back.trans.base.mat)
trellis.device(postscript,file="detecting_motion_test_bt_2.ps",
               width=6,height=6,horizontal=F)
plot.back.trans(tmp.back.trans,tmp.img,tmp.bg,tmp.fit.2$back.probs)
dev.off()


######
## plot histogram of new image:
trellis.device(postscript,file="detecting_motion_test_hist.ps",
               width=8,height=5,horizontal=F)
histogram(~tmp.img,breaks=seq(-0.5,255.5,by=7),
          xlab="Pixel reflective value",
          ylab="Density")
dev.off()

## veh. only:
tmp.veh.ind <- matrix(F,30,20)
tmp.veh.ind[6:12,4:8] <- T
tmp.veh.ind[18:25,12:17] <- T
trellis.device(postscript,file="detecting_motion_test_hist_veh.ps",
               width=8,height=5,horizontal=F)
histogram(~tmp.img[tmp.veh.ind],breaks=seq(-0.5,255.5,by=7),
          xlab="Pixel reflective value",
          ylab="Density")
dev.off()

#####
## create classification table for three methods:

## thresholding (use 5% vehicles, 70% in lower tail)
tmp <- order(tmp.img)
n <- length(tmp.img)
n.veh <- n*0.05
thresh.ind <- c(tmp[1:round(n.veh*0.7)],tmp[round(n-n.veh*0.3):n])
tmp <- table(tmp.veh.ind[thresh.ind])
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100)
```

```
## thresholding (use 15% vehicles, 70% in lower tail)
tmp <- order(tmp.img)
n <- length(tmp.img)
n.veh <- n*0.15
thresh.ind <- c(tmp[1:round(n.veh*0.7)],tmp[round(n-n.veh*0.3):n])
tmp <- table(tmp.veh.ind[thresh.ind])
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100)

## thresholding (use 25% vehicles, 70% in lower tail)
tmp <- order(tmp.img)
n <- length(tmp.img)
n.veh <- n*0.25
thresh.ind <- c(tmp[1:round(n.veh*0.7)],tmp[round(n-n.veh*0.3):n])
tmp <- table(tmp.veh.ind[thresh.ind])
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100)


####
## 1 parameter transformation:
## 5% traffic density at prior:
tmp.fit <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                              traffic.dens=0.05,
                              back.trans.control=list(nr.trans.param=1))
tmp <- tmp.veh.ind[tmp.fit$back.probs<0.5]
tmp <- table(tmp)
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100)

## 15% traffic density at prior:
tmp.fit <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                              traffic.dens=0.15,
                              back.trans.control=list(nr.trans.param=1))
tmp <- tmp.veh.ind[tmp.fit$back.probs<0.5]
tmp <- table(tmp)
cat("15% at prior\n")
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100)

## 25% traffic density at prior:
tmp.fit <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                              traffic.dens=0.25,
                              back.trans.control=list(nr.trans.param=1))
tmp <- tmp.veh.ind[tmp.fit$back.probs<0.5]
tmp <- table(tmp)
cat("25% at prior\n")
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100)


####
## 5 parameter transformation:
## 5% traffic density at prior:
tmp.fit <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                              traffic.dens=0.05,
                              back.trans.control=list(nr.trans.param=5))
tmp <- tmp.veh.ind[tmp.fit$back.probs<0.5]
tmp <- table(tmp)
cat("5 param, and 5% at prior\n")
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100)

## 15% traffic density at prior:
```

```
tmp.fit <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                              traffic.dens=0.15,
                              back.trans.control=list(nr.trans.param=5))
tmp <- tmp.veh.ind[tmp.fit$back.probs<0.5]
tmp <- table(tmp)
cat("5 param, and 15% at prior\n")
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100)

## 25% traffic density at prior:
tmp.fit <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                              traffic.dens=0.25,
                              back.trans.control=list(nr.trans.param=5))
tmp <- tmp.veh.ind[tmp.fit$back.probs<0.5]
tmp <- table(tmp)
cat("5 param, and 25% at prior\n")
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100)


#####
## use images A and B:(B image and A background)
## use 3% traffic density as prior
tmp.ind <- i70b.56.cut.ind
tmp.img <- i70b.57.bi[tmp.ind]
tmp.bg <- i70b.56.bg[tmp.ind]

## use 1 parameter model
tmp.fit <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                              traffic.dens=0.03,
                              back.trans.control=list(nr.trans.param=1))
## plot weights:
tmp <- i70b.57.bi
tmp[!tmp.ind] <- NA
tmp1 <- tmp
tmp1[tmp.ind] <- round(255*tmp.fit$back.probs)
image.device('postscript',file="detecting_motion_img_3pc_cc_1_2.ps",
             data.dim=dim(tmp1),height=6)
plot.image(tmp1)
dev.off()
tmp[tmp.ind] <- ifelse(tmp.fit$back.probs>=0.5,255,0)
image.device('postscript',file="detecting_motion_img_3pc_pp_1_2.ps",
             data.dim=dim(tmp),height=6)
plot.image(tmp)
dev.off()

tmp <- (i70b.57.veh.ind[tmp.ind])[tmp.fit$back.probs<0.5]
tmp <- table(tmp)
cat("1 param, and 3% at prior\n")
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100,
  'omission'=(sum(i70b.57.veh.ind[tmp.ind])-tmp['TRUE'])/sum(i70b.57.veh.ind[tmp.ind])*100)

## use 2 parameter model _ shift & slope
tmp.fit <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                              traffic.dens=0.03,
                              back.trans.control=list(nr.trans.param=2))
## plot weights:
tmp <- i70b.57.bi
tmp[!tmp.ind] <- NA
tmp1 <- tmp
tmp1[tmp.ind] <- round(255*tmp.fit$back.probs)
image.device('postscript',file="detecting_motion_img_3pc_cc_2_2.ps",
             data.dim=dim(tmp1),height=6)
plot.image(tmp1)
dev.off()
tmp[tmp.ind] <- ifelse(tmp.fit$back.probs>=0.5,255,0)
```

```
image.device('postscript',file="detecting_motion_img_3pc_pp_2_2.ps",
             data.dim=dim(tmp),height=6)
plot.image(tmp)
dev.off()

tmp <- (i70b.57.veh.ind[tmp.ind])[tmp.fit$back.probs<0.5]
tmp <- table(tmp)
cat("2 param, and 3% at prior\n")
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100,
  'omission'=(sum(i70b.57.veh.ind[tmp.ind])-tmp['TRUE'])/sum(i70b.57.veh.ind[tmp.ind])*100)

## use 5 parameter model
tmp.fit <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                              traffic.dens=0.03,
                              back.trans.control=list(nr.trans.param=5))
## plot weights:
tmp <- i70b.57.bi
tmp[!tmp.ind] <- NA
tmp1 <- tmp
tmp1[tmp.ind] <- round(255*tmp.fit$back.probs)
image.device('postscript',file="detecting_motion_img_3pc_cc_5_2.ps",
             data.dim=dim(tmp1),height=6)
plot.image(tmp1)
dev.off()

tmp[tmp.ind] <- ifelse(tmp.fit$back.probs>=0.5,255,0)
image.device('postscript',file="detecting_motion_img_3pc_pp_5_2.ps",
             data.dim=dim(tmp),height=6)
plot.image(tmp)
dev.off()

tmp <- (i70b.57.veh.ind[tmp.ind])[tmp.fit$back.probs<0.5]
tmp <- table(tmp)
cat("5 param, and 3% at prior\n")
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100,
  'omission'=(sum(i70b.57.veh.ind[tmp.ind])-tmp['TRUE'])/sum(i70b.57.veh.ind[tmp.ind])*100)

## thresholding:
tmp <- order(tmp.img)
n <- length(tmp.img)
n.veh <- n*0.03
thresh.ind <- c(tmp[1:round(n.veh*0.7)],tmp[round(n-n.veh*0.3):n])
tmp <- table((i70b.57.veh.ind[tmp.ind])[thresh.ind])
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100,
  'omission'=(sum(i70b.57.veh.ind[tmp.ind])-tmp['TRUE'])/sum(i70b.57.veh.ind[tmp.ind])*100)

tmp <- i70b.57.bi
tmp[] <- 255
tmp[tmp.ind][thresh.ind] <- 0
image.device('postscript',file="detecting_motion_img_3pc_thres_1_2.ps",
             data.dim=dim(tmp),height=6)
plot.image(tmp)
dev.off()


#####
## use images A and B:(B image and A background)
## use 1% traffic density as prior
tmp.ind <- i70b.56.cut.ind
tmp.img <- i70b.57.bi[tmp.ind]
tmp.bg <- i70b.56.bg[tmp.ind]

## use 1 parameter model
```

```
tmp.fit <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                              traffic.dens=0.01,
                              back.trans.control=list(nr.trans.param=1))
## plot weights:
tmp <- i70b.57.bi
tmp[!tmp.ind] <- NA
tmp1 <- tmp
tmp1[tmp.ind] <- round(255*tmp.fit$back.probs)
image.device('postscript',file="detecting_motion_img_1pc_cc_1_2.ps",
             data.dim=dim(tmp1),height=6)
plot.image(tmp1)
dev.off()
tmp[tmp.ind] <- ifelse(tmp.fit$back.probs>=0.5,255,0)
image.device('postscript',file="detecting_motion_img_1pc_pp_1_2.ps",
             data.dim=dim(tmp),height=6)
plot.image(tmp)
dev.off()

tmp <- (i70b.57.veh.ind[tmp.ind])[tmp.fit$back.probs<0.5]
tmp <- table(tmp)
cat("1 param, and 1% at prior\n")
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100,
  'omission'=(sum(i70b.57.veh.ind[tmp.ind])-tmp['TRUE'])/sum(i70b.57.veh.ind[tmp.ind])*100)

## use 2 parameter model _ shift & slope
tmp.fit <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                              traffic.dens=0.01,
                              back.trans.control=list(nr.trans.param=2))
## plot weights:
tmp <- i70b.57.bi
tmp[!tmp.ind] <- NA
tmp1 <- tmp
tmp1[tmp.ind] <- round(255*tmp.fit$back.probs)
image.device('postscript',file="detecting_motion_img_1pc_cc_2_2.ps",
             data.dim=dim(tmp1),height=6)
plot.image(tmp1)
dev.off()
tmp[tmp.ind] <- ifelse(tmp.fit$back.probs>=0.5,255,0)
image.device('postscript',file="detecting_motion_img_1pc_pp_2_2.ps",
             data.dim=dim(tmp),height=6)
plot.image(tmp)
dev.off()

tmp <- (i70b.57.veh.ind[tmp.ind])[tmp.fit$back.probs<0.5]
tmp <- table(tmp)
cat("2 param, and 1% at prior\n")
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100,
  'omission'=(sum(i70b.57.veh.ind[tmp.ind])-tmp['TRUE'])/sum(i70b.57.veh.ind[tmp.ind])*100)

## use 5 parameter model
tmp.fit <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                              traffic.dens=0.01,
                              back.trans.control=list(nr.trans.param=5))
## plot weights:
tmp <- i70b.57.bi
tmp[!tmp.ind] <- NA
tmp1 <- tmp
tmp1[tmp.ind] <- round(255*tmp.fit$back.probs)
image.device('postscript',file="detecting_motion_img_1pc_cc_5_2.ps",
             data.dim=dim(tmp1),height=6)
plot.image(tmp1)
dev.off()

tmp[tmp.ind] <- ifelse(tmp.fit$back.probs>=0.5,255,0)
image.device('postscript',file="detecting_motion_img_1pc_pp_5_2.ps",
             data.dim=dim(tmp),height=6)
```

```
plot.image(tmp)
dev.off()

tmp <- (i70b.57.veh.ind[tmp.ind])[tmp.fit$back.probs<0.5]
tmp <- table(tmp)
cat("5 param, and 1% at prior\n")
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100,
  'omission'=(sum(i70b.57.veh.ind[tmp.ind])-tmp['TRUE'])/sum(i70b.57.veh.ind[tmp.ind])*100)

## thresholding:
tmp <- order(tmp.img)
n <- length(tmp.img)
n.veh <- n*0.01
thresh.ind <- c(tmp[1:round(n.veh*0.7)],tmp[round(n-n.veh*0.3):n])
tmp <- table((i70b.57.veh.ind[tmp.ind])[thresh.ind])
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100,
  'omission'=(sum(i70b.57.veh.ind[tmp.ind])-tmp['TRUE'])/sum(i70b.57.veh.ind[tmp.ind])*100)

tmp <- i70b.57.bi
tmp[] <- 255
tmp[tmp.ind][thresh.ind] <- 0
image.device('postscript',file="detecting_motion_img_1pc_thres_1_2.ps",
             data.dim=dim(tmp),height=6)
plot.image(tmp)
dev.off()


#####
## use images A and B:(B image and A background)
## use 7% traffic density as prior
tmp.ind <- i70b.56.cut.ind
tmp.img <- i70b.57.bi[tmp.ind]
tmp.bg <- i70b.56.bg[tmp.ind]

## use 1 parameter model
tmp.fit <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                              traffic.dens=0.07,
                              back.trans.control=list(nr.trans.param=1))
## plot weights:
tmp <- i70b.57.bi
tmp[!tmp.ind] <- NA
tmp1 <- tmp
tmp1[tmp.ind] <- round(255*tmp.fit$back.probs)
image.device('postscript',file="detecting_motion_img_7pc_cc_1_2.ps",
             data.dim=dim(tmp1),height=6)
plot.image(tmp1)
dev.off()
tmp[tmp.ind] <- ifelse(tmp.fit$back.probs>=0.5,255,0)
image.device('postscript',file="detecting_motion_img_7pc_pp_1_2.ps",
             data.dim=dim(tmp),height=6)
plot.image(tmp)
dev.off()

tmp <- (i70b.57.veh.ind[tmp.ind])[tmp.fit$back.probs<0.5]
tmp <- table(tmp)
cat("1 param, and 7% at prior\n")
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100,
  'omission'=(sum(i70b.57.veh.ind[tmp.ind])-tmp['TRUE'])/sum(i70b.57.veh.ind[tmp.ind])*100)

## use 2 parameter model _ shift & slope
tmp.fit <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                              traffic.dens=0.07,
                              back.trans.control=list(nr.trans.param=2))
## plot weights:
```

```
tmp <- i70b.57.bi
tmp[!tmp.ind] <- NA
tmp1 <- tmp
tmp1[tmp.ind] <- round(255*tmp.fit$back.probs)
image.device('postscript',file="detecting_motion_img_7pc_cc_2_2.ps",
             data.dim=dim(tmp1),height=6)
plot.image(tmp1)
dev.off()
tmp[tmp.ind] <- ifelse(tmp.fit$back.probs>=0.5,255,0)
image.device('postscript',file="detecting_motion_img_7pc_pp_2_2.ps",
             data.dim=dim(tmp),height=6)
plot.image(tmp)
dev.off()

tmp <- (i70b.57.veh.ind[tmp.ind])[tmp.fit$back.probs<0.5]
tmp <- table(tmp)
cat("2 param, and 7% at prior\n")
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100,
  'omission'=(sum(i70b.57.veh.ind[tmp.ind])-tmp['TRUE'])/sum(i70b.57.veh.ind[tmp.ind])*100)

## use 5 parameter model
tmp.fit <- run.EM.norm.and.ns(tmp.img,tmp.bg,update.veh.dens=F,
                              traffic.dens=0.07,
                              back.trans.control=list(nr.trans.param=5))
## plot weights:
tmp <- i70b.57.bi
tmp[!tmp.ind] <- NA
tmp1 <- tmp
tmp1[tmp.ind] <- round(255*tmp.fit$back.probs)
image.device('postscript',file="detecting_motion_img_7pc_cc_5_2.ps",
             data.dim=dim(tmp1),height=6)
plot.image(tmp1)
dev.off()

tmp[tmp.ind] <- ifelse(tmp.fit$back.probs>=0.5,255,0)
image.device('postscript',file="detecting_motion_img_7pc_pp_5_2.ps",
             data.dim=dim(tmp),height=6)
plot.image(tmp)
dev.off()

tmp <- (i70b.57.veh.ind[tmp.ind])[tmp.fit$back.probs<0.5]
tmp <- table(tmp)
cat("5 param, and 7% at prior\n")
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100,
  'omission'=(sum(i70b.57.veh.ind[tmp.ind])-tmp['TRUE'])/sum(i70b.57.veh.ind[tmp.ind])*100)

## thresholding:
tmp <- order(tmp.img)
n <- length(tmp.img)
n.veh <- n*0.07
thresh.ind <- c(tmp[1:round(n.veh*0.7)],tmp[round(n-n.veh*0.3):n])
tmp <- table((i70b.57.veh.ind[tmp.ind])[thresh.ind])
tmp
c('total'=sum(tmp),'correcr'=tmp['TRUE']/sum(tmp)*100,
  'wrong'=tmp['FALSE']/sum(tmp)*100,
  'omission'=(sum(i70b.57.veh.ind[tmp.ind])-tmp['TRUE'])/sum(i70b.57.veh.ind[tmp.ind])*100)

tmp <- i70b.57.bi
tmp[] <- 255
tmp[tmp.ind][thresh.ind] <- 0
image.device('postscript',file="detecting_motion_img_7pc_thres_1_2.ps",
             data.dim=dim(tmp),height=6)
plot.image(tmp)
dev.off()
```

4.3 The S+ Image Processing Code

```
###########              ################################################
# GJ: 5-NOV-97
#
# Collection of functions to deal with gray-scale images.
#
#
#####          ##              ########################################

# function to read in Images in ASCII format.
# Returns a matrix

imagine.2.s <- function(file,compresed=T) {
  if(compresed) {
    tmp.file <- tempfile()
    unix(paste("uncompress -c ",file," > ",tmp.file,sep=""))
    on.exit(unix(paste("rm -f",tmp.file)))
    file <- tmp.file
  }
  data <- matrix(scan(file,skip=4),byrow=T,ncol=3) # (x,y,z) data
  ux <- sort(unique(data[,1]))
  uy <- sort(unique(data[,2]))
  data <- matrix(data[,3],byrow=T,nrow=length(uy),ncol=length(ux),
                dimnames=list(uy,ux))
  attr(data,"header") <- scan(file,n=3,what="")[3]
  return(data)
}

##########################################################################

# edge detection -- gradiant method on 3x3 mask

detect.edge <- function(data) {

  dd <- dim(data)
  n.na <- 0
  search.na <- T
  while(search.na) {
    search.na <- all(is.na(data[,n.na+1]))
    n.na <- n.na + search.na
  }
  print(n.na)
  x.r <- (n.na+2):(dd[2]-n.na-1)
  y.r <- (n.na+2):(dd[1]-n.na-1)
  # in the x-direction:
  data.x <- 2*data[y.r,] + data[y.r-1,] + data[y.r+1,]
  x.grad <- data.x[,x.r+1] - data.x[,x.r-1]
  data.y <- 2*data[,x.r] + data[,x.r-1] + data[,x.r+1]
  y.grad <- data.y[y.r-1,] - data.y[y.r+1,]

  size <- angle <- matrix(NA,nrow=dd[1],ncol=dd[2])
  angle[y.r,x.r] <- atan(y.grad/x.grad)
  size[y.r,x.r] <- sqrt(y.grad^2+x.grad^2)

  return(size=size,angle=angle)
}

##########################################################################

apply.filter <- function(data,weights=rbind(c(1,2,1),c(2,4,2),c(1,2,1)),
                         n.na=0) {

  # the weights are given row by row.
  # by default it is a 'binomial' mask.

  weights <- weights/sum(weights)
  n <- length(weights)
  dd <- dim(data)
```

```
  dw <- dim(weights)
  no.na.s <- !is.na(data)
  data[!no.na.s] <- 0

  x.r <- 1:(dd[2]-2*(n.na+1))
  y.r <- 1:(dd[1]-2*(n.na+1))
  result <- total.weights <- matrix(0,nrow=dd[1],ncol=dd[2])
  for(i in 1:nrow(weights))
    for(j in 1:ncol(weights)) {
      result[y.r+n.na+1,x.r+n.na+1] <- result[y.r+n.na+1,x.r+n.na+1] +
        weights[i,j]*data[y.r-1+i,x.r-1+j]
    total.weights[y.r+n.na+1,x.r+n.na+1] <-
       total.weights[y.r+n.na+1,x.r+n.na+1] + no.na.s[y.r-1+i,x.r-1+j]
    }
  result <- result*(length(weights)/total.weights)
  return(result)
}


####       ################################################################

image.device <- function(device=c("motif","postscript"),file="image.ps",
                         height=10.5,width=8,dpi,n.colors=256,data.dim=NULL,
                         horizontal=F,...) {


  device <- match.arg(device)
  assign("greylevels.256colors",seq(0,1,le=n.colors),where=0)
  ps.options(colors=greylevels.256colors,background=-1)

  if(device=="motif") {
    add.to.sgraphrc <- "-xrm 'sgraphMotif.colorSchemes: name: \"256
    greylevels\ "; background: white; lines: black h5 white; text:
    black h5 white; polygons: black h254 white; images: black h254
    white'"
    motif(options=add.to.sgraphrc,...)
  } else {
    # a postscript file is created just to suround the image.
    #figure out the size of the postscript file:
    if(!is.null(data.dim)) {
      d.ratio <- data.dim[1]/data.dim[2]
      p.ratio <- height/width
      if(d.ratio>=p.ratio)
        width <- height*(1/d.ratio)
      else
        height <- width*d.ratio
    }
    postscript(file=file,width=width,height=height,horizontal=horizontal,
               onefile=F,print.it=F,
               colors=greylevels.256colors,image.colors=greylevels.256colors)
  }

  par(xaxs="i",yaxs="i")
  par(mar=c(0,0,0,0))

  return(invisible())
}

##########################################################################

scale.image <- function(data,n.colors=256,reverse=F) {

  ind <- is.na(data)  #background
  d.r <- range(data[!ind])
  data <- round((data-d.r[1])/(d.r[2]-d.r[1]) * (n.colors-1))
  if(reverse)
    data <- (n.colors-1)-data
  data[ind] <- NA
  return(data)
}
```

```
##########################################################################

plot.image <- function(data,add=F,n.colors=256,add.grid=F,add.frame=T,
                       method=c("image","polygon")) {
  # data is a matrix with gray-scale values.

  dd <- dim(data)
  n.row <- dd[1]; n.col <- dd[2]
  d.ratio <- n.row/n.col

  if(!add)
    par(pin=par()$din)

  p.par <- par()$pin
  p.ratio <- p.par[2]/p.par[1]
  if(d.ratio >= p.ratio) {
    par(pin=c(p.par[2]/d.ratio,p.par[2]))
  }
  if(d.ratio < p.ratio) {
    par(pin=c(p.par[1],p.par[1]*d.ratio))
  }

  if(!add)
    plot(c(0,n.col)+0.5,c(0,n.row)+0.5,type="p",
         xlab="",ylab="",axes=F,xaxs="i",yaxs="i",col=0)

  par(err=-1)

  ux <- seq(0.5,n.col+0.5,by=1)
  uy <- seq(n.row+0.5,0.5,by=-1)
  method <- match.arg(method)
  if(method=="polygon") {
    data[is.na(data)] <- -1  # the background
    .C("polygon_matrix",
       as.single(ux),
       as.integer(length(ux)),
       as.single(uy),
       as.integer(length(uy)),
       as.single(c(0,1:n.colors)[data+2])
       )
  } else {
    image(x=ux,y=uy,z=t(data)+1,add=add)
  }

  if(add.grid) {
    abline(v=ux,lwd=0.5,lty=1)
    abline(h=uy,lwd=0.5,lty=1)
  }
  if(add.frame) {
    abline(v=range(ux),lwd=0.5,lty=1)
    abline(h=range(uy),lwd=0.5,lty=1)
  }

  return(invisible())
}

                       ##################################################
```

Appendix C. Log-Normal and Poisson Traffic Count Data Simulation Programs

Documentation for Traffic Count Simulation_Log-Normal Errors: v2.0A

****

This program simulates a network of road links that are sampled by satellite photos and ATRs. The data are generated according to a log-linear model with normal errors. The segment lengths must be supplied in the file length. Expansion factors are now read from the file 'expfactor.in'

****

V LES

MAXLINK = maximum number of links possible *** It's the dimension of linkMean()***
idum = random number seed used by ran1() and gasdev() ***Using the same idum gives the same output***
nlink = actual number of links used
nsat = number of sats
natr = number of perm ATR
nportatr = number of moveable ATR
link_mean(i) = AADT of link i
link_length(i) = length of link i
mini = minimum AADT e.g. 10,000
maxi = maximum AADT eg 90,000
hef(24) = hourly expansion factor. Not currently used.
def(7) = day of week expansion factor
mef(12) = monthly expansion factor
dt = #days from one satellite overpass to another
coverage = proportion of links seen by satellite for one overpass ***e.g., coverage = 0.01 = 1% of links seen***
timeint = effective length of time (in hours) of traffic "seen" by satellite. ***e.g., timeint = 0.0167 => satellite will count a minutes worth of traffic. Not currently used.***
sigma = variability of counts. *** e.g., sigma = 0.10 => 10% variability in recorded count***
There are two sigmas used: sigmasat, sigmaground

LIST OF MODULES IN PROGRAM:

/* Read expansion factors from file 'expfactor.in' */

/* Read random seed from file 'idum.in' */

/* Read input file and write to some parameter files */

/* Get and prepare link lengths, write to file */

/* Generate EF and write to files */

/* Generate link parameters and write to files */

/* Generate satellite data and write to files */

/* Generate cts ATRs. The links are 0,...,natr-1, so the link lengths for the cts ATRs are always the same. Write data to a file. */

/* generate short term ATRs and write data to a file */

SUBRO USED:

/* readseed: return the random seed from file idum.in. The random seed is a negative integer. */

/* read_EF: read the seasonal adjustment factors */

/* read_input: read file 'input' for parameters */
/* Read number of links, nlink */
/* Read UB and LB on link AADT */
/* Read sat parameters */
/* Read cts ATR parameters */
/* Enter portable ATR parameters */

/* get_lengths: read and prepare the link lengths as follows:
Lengths are read from file length.in
The first natr are always assigned to the PATR, so that the PATR always have the same link lengths.
Finally the remaining nlink - natr lengths are assigned randomly to the links w/o PATR, so the MATR are assigned to random links.
Lengths are written to length.out */
/* Read the lengths from 'length.in'*/
/* Scramble the last nlink - natr links */
/* Write the results to file */

/* gen_EFO: generate expansion factors and write to file 'truth.out'. */

/* gen_link_par : Generate linkMean, linkLength, total traffic, AADT, VMT. Write above to files. */
/* Generate true mean of daily traffic count for each link */
/* Write out total volume of traffic for the year to 'truth.out'. */
/* Write link_mean (AADT of links) to files 'truth.out' and 'aadt.out' */
/* Compute VMT and write to file 'vmt.out' */

/* gen_sat: write simulated sat counts to file 'sate.out' and write sampling design to file 'design.out'. Write link and number of times each link is sampled by sat to file 'sat_samp.out' */
/* Initialize satsamp */
/* Sample from sat */

/* gen_ATR: generate ATR counts */

/* ran1: generate a realization of a uniform(0,1) rv */

/* gasdev: return a realization of a std normal rv */

/* get_sample: put sample of size n into first n slots of linkID 0 to n-1.
These links are sampled by the sat on a given pass, or used by the MATR.
The first excl links are excluded. Excl = npatr for MATR, or 0 for sat */

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include <time.h>

#define MAXLINK 5000  /* Maximum number of links allowed */


/* Function prototypes. */

int readseed(void);
void read_EF(double hef[], double def[], double mef[]);
void get_lengths(double link_length[], int nlink, int natr, int maxlink,
                 int *idum);
double ran1(int *idum);
double gasdev(int *idum);
void gen_EFO(double hef[], double def[], double mef[]);
void gen_link_parameters(double link_mean[], double link_length[],
                         int maxlink, int nlink, double mini, double maxi,
                         int *idum);
void gen_sat(double link_mean[], int maxlink, int nlink, double dt,
             double def[], double mef[], double coverage, double sigma,
             int *idum);
void gen_ATR(double link_mean[], int maxlink, int nlink, int link,
             int yearday1, int yearday2, double hef[], double def[],
             double mef[], double sigmaground, int *idum);
void read_input(int *nlink, int *nsat, double *dt, double *coverage,
                double *timeint, double *sigmasat, int *natr,
                double *sigmaground, int *nportatr, double *mini,
                double *maxi);
void get_sample(int linkID[], int n, int maxlink, int nlink, int *idum,
                int excl);
void convertDayNumber(double time, double *hh, int *dd, int *mm,
                      int *weekday);
double max(double a, double b);
double min(double a, double b);

/****************************************************************************/

main()
{
 /* Declare variables: */

  int i, idum, natr, link, yearday1, yearday2, length, start, nportatr,
    nlink, nsat = 0, excl;
  double link_mean[MAXLINK], link_length[MAXLINK];
  double dt, coverage, timeint, sigmasat, sigmaground, mini, maxi;
  double hef[24], def[7], mef[12];
```

```
/* Read expansion factors from file 'expfactor.in' */

read_EF(hef, def, mef);

/* Read random seed from file 'idum.in' */
idum = readseed();

/* Read input file and write to some parameter files */
read_input(&nlink, &nsat, &dt, &coverage, &timeint, &sigmasat, &natr,
           &sigmaground, &nportatr, &mini, &maxi);

/* Get and prepare link lengths, write to file */
get_lengths(link_length, nlink, natr, MAXLINK, &idum);

/* Generate EF and write to files */
gen_EFO(hef, def, mef);

/* Generate link parameters and write to files */
gen_link_parameters(link_mean, link_length, MAXLINK, nlink, maxi, mini,
                    &idum);

/* Generate satellite data and write to files */
for (i = 1; i <= nsat; i++) {
  gen_sat(link_mean, MAXLINK, nlink, dt, def, mef, coverage, sigmasat,
          &idum);
}

/* Generate cts ATRs.  The links are 0,...,natr-1, so the link lengths for
   the cts ATRs are always the same. */
yearday1 = 1;
yearday2 = 365;

for (i = 0; i < natr; i++) {
  link = i;
  gen_ATR(link_mean, MAXLINK, nlink, link, yearday1, yearday2, hef, def, mef,
          sigmaground, &idum);
}

/* generate short term ATRs */
for (i = natr; i < natr + nportatr; i++) {
  length = 2; /* Number of days of observations at any MATR link. */
  start = floor( ran1(&idum) * (365-length+1) ) + 1;
  gen_ATR(link_mean, MAXLINK, nlink, i, start, start+length-1, hef, def,
          mef, sigmaground, &idum);
}

return 0;
} /* End of main */


/* Function definitions: */
```

```
/* readseed: return the random seed from file idum.in.  The random seed is a
   negative integer. */
int readseed(void)
{
  int c = 0;
  FILE *idump;
  idump = fopen("idum.in", "r");
  fscanf(idump, "%d", &c);
  fclose(idump);
  if (!(c < 0))
    printf("\nreadseed: error, random seed must be a negative integer.\n");
  return c;
}

/* read_EF: read the seasonal adjustment factors */
void read_EF(double hef[], double def[], double mef[])
{
  int i;

  FILE *EFp;

  EFp = fopen("expfactor.in", "r");

  for (i = 0; i < 7; i++)
    fscanf(EFp, "%lf", &def[i]);
  for (i = 0; i < 12; i++)
    fscanf(EFp, "%lf", &mef[i]);

  fclose(EFp);

  return;
}


/* read_input: read file 'input' for parameters */
void read_input(int *nlink, int *nsat, double *dt, double *coverage,
         double *timeint, double *sigmasat, int *natr,
         double *sigmaground, int *nportatr, double *mini,
                 double *maxi)
{

  int i;

  FILE *parametersp;
  FILE *truthp;
  FILE *patrp;
  FILE *matrp;

  matrp = fopen("matr.out", "w");
  patrp = fopen("patr.out", "w");
  truthp = fopen("truth.out", "w");
```

```
parametersp = fopen("parameters.out", "w");

/* Read number of links, nlink */
*nlink = 0;
do {
  /* printf("Enter number of links for this run.\n"); */
  scanf("%d", nlink);
} while (*nlink < 1);
fprintf(truthp, "\nThere are %d links for this run.\n", *nlink);
fprintf(parametersp, "There are %d links.\n\n", *nlink);

/* Read UB and LB on link AADT */
*mini = 0;
do {
  /*  printf("Input lower bound for link AADT (min 1.0): \n"); */
  scanf("%lf", mini);
} while (*mini < 1);

*maxi = *mini;
do {
  /* printf("Input upper bound for link AADT: \n"); */
  scanf("%lf", maxi);
} while (*maxi <= *mini);

fprintf(truthp, "Lower bound of AADT = %f. Upper bound = %f\n", *mini,*maxi);
fprintf(parametersp, "Bounds are from %f to %f.\n", *mini, *maxi);


/* read sat data */

*nsat = -1, *dt = 0, *coverage = -1;

do {
  /* printf("Enter number of satellites: \n"); */
  scanf("%d", nsat);
} while (*nsat < 0);
*nsat = ( *nsat > *nlink) ? *nlink : *nsat; /* Truncate nsat at nlink */
fprintf(parametersp, "There are %d satellites.\n", *nsat);

do {
  /* printf("Input time between sat passes, in days.\n"); */
  scanf("%lf", dt);
} while (*dt < .01);
fprintf(parametersp, "Time between sat passes = %f days.\n", *dt);

do {
  /* printf("Input fraction of links seen by satellite.\n"); */
  scanf("%lf", coverage);
} while (*coverage < 0 || *coverage > 1);
fprintf(parametersp, "Coverage = %f percent.\n", *coverage * 100);

/* printf("Input fraction of hour equivalent the sat sees.\n");
```

```
  printf("Be sure to make the value between .001 and 24.0\n"); */
 scanf("%lf", timeint);
 fprintf(parametersp, "Equivalent time = %f.\n", *timeint);

 /*printf("Input s > 0, where the error has exp( Normal[0, s^2] ) dist\n");
  printf("and s < 1 say.  s is sigma_sat.\n"); */
 *sigmasat = -1;
 do {
  scanf("%lf", sigmasat);
 } while  (*sigmasat < 0);
 fprintf(parametersp, "Sigmasat = %f.\n", *sigmasat);


 /* Enter cts ATR parameters */

 /* printf("Enter number of continuous ATRs, not greater than #links.\n");*/
 scanf("%d", natr);
 fprintf(parametersp, "There are %d continuous ATR links.\n", *natr);
 fprintf(patrp, "%5d\n", *natr);

 for (i = 0; i < *natr; i++)
  fprintf(patrp, "%8d\n", i+1);

 /* printf("Enter nonnegative sigma value for ground counts.\n"); */
 scanf("%lf", sigmaground);

 fprintf(parametersp, "sigmaground = %f\n", *sigmaground);


 /* Enter portable ATR parameters */

 scanf("%d", nportatr);
 fprintf(parametersp, "There are %d portable ATRs used.\n", *nportatr);
 fprintf(matrp, " %d\n", *nportatr);

 for (i = *natr; i < *natr + *nportatr; i++)
  fprintf(matrp, "%5d\n", i+1);

 fclose(matrp);
 fclose(truthp);
 fclose(parametersp);
 fclose(patrp);

 return;
}


/* get_lengths: read and prepare the link lengths as follows:
  Lengths are read from file length.in
  The first natr are always assigned to the PATR, so that the PATR
  always have the same link lengths.
  Finally the remaining nlink - natr lengths are assigned randomly
```

```
 to the links w/o PATR, so the MATR are assigned to random links.
 Lengths are written to length.out */
void get_lengths(double link_length[], int nlink, int natr, int maxlink,
                 int *idum)
{

 int i;
 int linkID[MAXLINK];
 double garb_dbl[MAXLINK];

 FILE *length_inp;
 FILE *lengthp;

 length_inp = fopen("length.in", "r");
 lengthp = fopen("length.out", "w");

 if (length_inp == NULL)
  printf( "read_lengths: file length.in not found");

 /* Read the lengths from 'length.in'*/
 for (i = 0; i < nlink; i++)
  fscanf(length_inp, "%lf", &link_length[i]);


/* Scramble the last nlink - natr links */

 get_sample(linkID, nlink - natr, MAXLINK, nlink, idum, natr);

 for (i = 0; i < nlink; i++)
  garb_dbl[i] = link_length[i];

 for (i = natr; i < nlink; i++)
  link_length[i] = garb_dbl[ linkID[i] ];


 /* Write the results to file */
 for (i = 0; i < nlink; i++)
  fprintf(lengthp, "%5d %7.4f\n", i+1, link_length[i]);

 fclose(length_inp);
 fclose(lengthp);

 return;
}

/* gen_EFO: generate expansion factors and write to file 'truth.out'. */
void gen_EFO(double hef[], double def[], double mef[])
{
 int i;
 double sum = 0.0;

 FILE *truthp;
```

```
truthp = fopen("truth.out", "a");

for (i = 0; i <= 22; i++)
  sum += 1.0 / hef[i];
hef[23] = 1.0 / (24.0 - sum);

sum = 0.0;
for (i = 0; i <= 5; i++)
  sum += 1.0 / def[i];
def[6] = 1.0 / (7.0 - sum);

sum = 0.0;
for (i = 0; i <= 10; i++)
  sum += 1.0 / mef[i];
mef[11] = 1.0 / (12.0 - sum);

fprintf(truthp, "Hourly expansion factors\n");
for (i = 0; i < 24; i++)
  fprintf(truthp, "From %d to %d, E.F. = %f\n", i, i+1, hef[i]);

fprintf(truthp, "\nWeekday expansion factors\n");
for (i = 0; i < 7; i++)
  fprintf(truthp, "From %d to %d, E.F. = %f\n", i, i+1, def[i]);

fprintf(truthp, "\nMonthly expansion factors\n");
for (i = 0; i < 12; i++)
  fprintf(truthp, "From %d to %d, E.F. = %f\n", i, i+1, mef[i]);

fclose(truthp);
return;
}


/* gen_link_parameters: Generate linkMean, linkLength, total traffic, AADT,
 VMT.  Write above to files. */
void gen_link_parameters(double link_mean[], double link_length[],
                         int maxlink, int nlink, double mini, double maxi,
                         int *idum)
{
 double sum = 0, adjust, proposed, mean_length, sd_length, min_length, temp,
   timeint;
 int i, count;

 FILE *truthp;
 FILE *aadtp;
 FILE *vmtp;

 truthp = fopen("truth.out", "a");
 aadtp = fopen("aadt.out", "w");
 vmtp = fopen("vmt.out", "w");
```

```
/* Generate true mean of daily traffic count for each link, then adjust to
   ensure that the total traffic is (min+max)/2.0 */
sum = 0.0;
for (i = 0; i < nlink; i++) {
  temp = ran1(idum);
  link_mean[i] = temp * (maxi - mini) + mini;
  sum += link_mean[i];
}

adjust = (float)sum / nlink - (mini + maxi) / 2.0;
for (i = 0; i < nlink; i++)
  link_mean[i] -= adjust;

/* Write out total volume of traffic for the year to 'truth.out'. */
sum = 0.0;
for (i = 0; i < nlink; i++)
  sum += link_mean[i];
fprintf(truthp, "Total volume of traffic for year, all links = %.0f\n\n",
          365*sum);

/* Write link_mean (AADT of links) to files 'truth.out' and 'aadt.out' */
sum = 0.0;
for (i = 0; i < nlink; i++) {
  sum += link_mean[i];
  fprintf(truthp, "Link %d has true AADT = %12.4f\n", i+1 , link_mean[i]);
  fprintf(aadtp, "%12.4f   %d\n", link_mean[i], i+1);
}
fprintf(truthp, "\nAverage AADT over all %d links = %12.4f\n", nlink,
          sum/(nlink));

/* Compute VMT and write to file 'vmt.out' */
sum = 0;
for (i = 0; i < nlink; i++)
  sum += link_mean[i] * link_length[i];
fprintf(vmtp, " %.0f.", sum);

fclose(vmtp);
fclose(aadtp);
fclose(truthp);

return;
}


/* gen_sat: write simulated sat counts to file 'sate.out' and write sampling
   design to file 'design.out'. Write link and number of times each link
   is sampled by sat to file 'sat_samp.out' */
void gen_sat(double link_mean[], int maxlink, int nlink, double dt,
             double def[], double mef[], double coverage, double sigma,
             int *idum)
{
  int n, i, mm, dd, count, weekday, excl;
```

```
int linkID[MAXLINK], satsamp[MAXLINK];
double time, AADT, rcount, hh;

FILE *parametersp;
FILE *satep;
FILE *designp;
FILE *satsampp;

parametersp = fopen("parameters.out","a");
satep = fopen("sate.out", "w");
designp = fopen("design.out", "w");
satsampp = fopen("sat_samp.out", "w");

n = ceil(coverage * nlink); /* Number of links sampled. */

fprintf(parametersp, "Number of links seen by sat is %d\n", n);

/* Initialize satsamp */
for (i = 0; i < nlink; i++)
  satsamp[i] = 0;


/* Sample from sat */

time = ran1(idum) * dt + 1;
excl = 0;
while (time < 366) {
  get_sample(linkID, n, MAXLINK, nlink, idum, excl);
  for (i = 0; i < n; i++) {
    AADT = link_mean[ linkID[i] ];
    convertDayNumber(time, &hh, &dd, &mm, &weekday);
    if (hh >= -1 && hh <= 25) { /* daytime: always for now */
        rcount = AADT / (def[weekday-1] * mef[mm-1]);
        rcount = rcount * exp( gasdev(idum) * sigma );
        rcount = rcount / exp( sigma * sigma / 2 ); /* bias correction */
        count = floor(rcount);
        fprintf(satep, "%5d %10d %3d %3d %2d\n", linkID[i]+1, count, mm, dd,
                weekday);
        fprintf(designp, "%10d %5d %2d %3d\n", count, linkID[i]+1,
                weekday, mm);

        (satsamp[ linkID[i] ])++;
    }
  }
  time = time + dt;
}

for (i = 0; i < nlink; i++)
  fprintf(satsampp, "%d %d\n", i+1, satsamp[i]);

fclose(designp);
fclose(parametersp);
```

```
 fclose(satep);
 fclose(satsampp);

 return;
}


/* gen_ATR: generate ATR counts */
void gen_ATR(double link_mean[], int maxlink, int nlink, int link,
        int yearday1, int yearday2, double hef[], double def[],
        double mef[], double sigmaground, int *idum)
{
 int dd, mm, weekday, i, count, isum = 0;
 double AADT, rcount, sum1, sumt, adjust, hh;
 double temp[365];

 FILE *patrp;
 FILE *matrp;
 FILE *designp;

 matrp = fopen("matr.out", "a");
 designp = fopen("design.out", "a");
 patrp = fopen("patr.out", "a");

 /* Sample from ATRs */

 if (yearday1 != 1 || yearday2 != 365) { /* movable atr */
  for (i = yearday1; i <= yearday2; i++) {
   AADT = link_mean[link];
   convertDayNumber((double)i, &hh, &dd, &mm, &weekday);
   rcount = AADT / (def[weekday-1] * mef[mm-1]);
   rcount = rcount * exp( gasdev(idum) * sigmaground );
   rcount = rcount / exp( sigmaground * sigmaground / 2 );
   count = floor(rcount);
   isum = isum + count;
   fprintf(designp, "%10d %5d %2d %3d\n", count, link+1, weekday, mm);
   fprintf(matrp, "%5d %10d %3d %3d %2d\n", link+1, count, mm, dd,
          weekday);
  }
 } else { /* permanent atr */
   sum1 = 0;
   sumt = 0;
   for ( i = yearday1; i <= yearday2; i++) {
       AADT = link_mean[link];
       sum1 += AADT;
       convertDayNumber((double)i, &hh, &dd, &mm, &weekday);
       rcount =AADT / (def[weekday-1] * mef[mm-1]);
       rcount = rcount * exp( gasdev(idum) * sigmaground );
       rcount = rcount / exp( sigmaground * sigmaground / 2 );
       temp[i-1] = floor(rcount);
       sumt += temp[i-1];
   }
```

```
    adjust = (sum1 - sumt) / (yearday2 - yearday1 + 1);
    sumt = 0;
    for ( i = yearday1; i <= yearday2; i++) {
        temp[i-1] += adjust;
        sumt += temp[i-1];
        count = floor(temp[i-1]);
        isum += count;
        convertDayNumber((double)i, &hh, &dd, &mm, &weekday);
        fprintf(patrp, "%5d %10d %3d %3d %2d\n", link+1, count, mm, dd,
               weekday);
        fprintf(designp, "%10d %5d %2d %3d\n", count, link+1, weekday, mm);
    }
   }

  fclose(patrp);
  fclose(designp);
  fclose(matrp);
  return;
}


/* ran1: generate a realization of a uniform(0,1) rv */
double ran1(int *idum)
{
  int ia=16807, im=2147483647, iq=127773, ir=2836, ntab=32, ndiv, j, k;
  double r1, am, eps, rnmx;
  /* next two should be static or something */
  static int iy ;
  static int iv[32]; /* dim is NTAB */
  /* statics are initialized to zero */

  am = 1 / (double)im;
  ndiv = 1 + (im - 1) / (double)ntab;
  eps =.         12;
  rnmx = 1 - eps;

  if (*idum<=0 || iy==0) {
   *idum = (int)max((double)(-(*idum)), 1.0);
   for (j = ntab+8; j >= 1; j--) {
    k = *idum / (double)iq;
    *idum = ia * (*idum - k*iq) - ir*k;
    if (*idum < 0)
        *idum += im;
    if (j <= ntab)
        iv[j] = *idum;
   }
   iy = iv[1];
  }

  k = *idum / (double)iq;
  *idum = ia * (*idum - k*iq) - ir*k;
  if (*idum < 0)
```

```
    *idum += im;
  j = 1 + iy / ndiv;
  iy = iv[j];
  iv[j] = *idum;

  r1 = min(am*iy, rnmx);
  return r1;

}


/* gasdev: return a realization of a std normal rv */
double gasdev(int *idum)
{
  static int iset;
  double fac, rsq, v1, v2, gdev;
  static double gset;

  if (iset == 0) {
  one:
    v1 = 2 * ran1(idum) - 1;
    v2 = 2 * ran1(idum) - 1;
    rsq = v1*v1 + v2*v2;
    if (rsq >= 1 || rsq == 0)
      goto one;
    fac = sqrt(-2 * log(rsq)/rsq);
    gset = v1 * fac;
    gdev = v2 * fac;
    iset = 1;
  }
  else {
    gdev = gset;
    iset = 0;
  }

  return gdev;
}

/* get_sample: put sample of size n into first n slots of linkID 0 to n-1.
   These links are sampled by the sat on a given pass, or used by the MATR.
   The first excl links are excluded.  Excl = npatr for MATR, or 0 for sat */
void get_sample(int linkID[], int n, int maxlink, int nlink, int *idum,
                        int excl)
{
  int i, k, num, temp;
  double rtemp;

  /* Initialize link IDs */
  for (i = 0; i < nlink; i++)
    linkID[i] = i;

  for (i = excl; i < n+excl; i++) {
```

```
    rtemp = ran1(idum) * (nlink-i) + i; /* a number in i to nlink */
    num = floor(rtemp); /* truncate so in i to nlink - 1 */

    temp = linkID[num];
    linkID[num] = linkID[i];
    linkID[i] = temp;
  }

  return;
}

/* convertDayNumber: */
void convertDayNumber(double time, double *hh, int *dd, int *mm,
                int *weekday)
{
  int yearday;
  double fraction;

  fraction = time - floor(time);
  yearday = floor(time - fraction);
  *weekday = yearday % 7 + 1;

  *hh = floor((time - yearday)*24) + 1;

  if (yearday <= 31 && yearday >=1) {
    *mm = 1;
    *dd = yearday;
  }

  if (yearday <= 59 && yearday >= 32) {
    *mm = 2;
    *dd = yearday - 31;
  }

  if (yearday <= 90 && yearday >= 60) {
    *mm = 3;
    *dd = yearday - 59;
  }

  if (yearday <= 120 && yearday >= 91) {
    *mm = 4;
    *dd = yearday - 90;
  }

  if (yearday <= 151 && yearday >= 121) {
    *mm = 5;
    *dd = yearday - 120;
  }

  if (yearday <= 181 && yearday >= 152) {
    *mm = 6;
    *dd = yearday - 151;
```

```
}

if (yearday <= 212 && yearday >= 182) {
  *mm = 7;
  *dd = yearday - 181;
}

if (yearday <= 243 && yearday >= 213) {
  *mm = 8;
  *dd = yearday - 212;
}

if (yearday <= 273 && yearday >= 244) {
  *mm = 9;
  *dd = yearday - 243;
}

if (yearday <= 304 && yearday >= 274) {
  *mm = 10;
  *dd = yearday - 273; }

if (yearday <= 334 && yearday >= 305) {
  *mm = 11;
  *dd = yearday - 304; }

if (yearday <= 365 && yearday >= 335) {
  *mm = 12;
  *dd = yearday - 334;
}

return;
}

/* max */
double max(double a, double b)
{
  double temp;

  temp =  (a > b) ? a : b;
  return temp;
}

/* min */
double min(double a, double b)
{
  double temp;

  temp = ( a < b) ? a : b;
  return temp;
}
```

B2: LISTING OF POISSON SIMULATION PROGRAM:

```
******************************************************************************************
THIS PROGRAM IS WRITEN IN S-PLUS
******************************************************************************************
*
function(seed, obs.params)
{
        set.seed(seed)     #DF and MF contain the appropriate daily and monthly
# factors for each of the 365 days of the year
        DF <- rep(DEF, 53)[1:365]
        MF <- c(rep(MEF[1], 31), rep(MEF[2], 28), rep(MEF[3], 31), rep(MEF[4],
                30), rep(MEF[5], 31), rep(MEF[6], 30), rep(MEF[7], 31), rep(MEF[
                8], 31), rep(MEF[9], 30), rep(MEF[10], 31), rep(MEF[11], 30),
                rep(MEF[12], 31))
        EF <- DF * MF   #read link parameters from link.params:
#n. of links; alpha and beta for the gamma prior;
        nlink <- obs.params[1]
        alpha <- obs.params[2]
        beta <- obs.params[3]
        nsat <- obs.params[4]
        repeatcycle <- obs.params[5]
        satcovg <- obs.params[6]
        npatr <- obs.params[7]
        nmatr <- obs.params[8]
        capacity <- obs.params[9]
        nsatdays <- (365 %/% repeatcycle) * nsat
        nsatobsday <- satcovg * nlink         #misc objects
        days <- seq(1, 365)
        evendays <- seq(2, 364, 2)
        links <- seq(npatr + 1, nlink)
        monthofday <- c(rep(1, 31), rep(2, 28), rep(3, 31), rep(4, 30), rep(5,
                31), rep(6, 30), rep(7, 31), rep(8, 31), rep(9, 30), rep(10, 31
                ), rep(11, 30), rep(12, 31))
        dateofday <- c(1:31, 1:28, 1:31, 1:30, 1:31, 1:30, 1:31, 1:31, 1:30, 1:
                31, 1:30, 1:31)     #generate link means
        theta <- beta * rgamma(nlink, alpha) + 10000            #generate PATR counts
        adjpatr <- matrix(nrow = 365 * npatr, ncol = 5)
        for(j in 1:npatr) {
                for(i in 1:365) {
                        adjpatr[i + (j - 1) * 365, 1] <- j
                        adjpatr[i + (j - 1) * 365, 2] <- rpois(n = 1, theta[j]/
                                EF[i])
                        adjpatr[i + (j - 1) * 365, 3] <- monthofday[i]
                        adjpatr[i + (j - 1) * 365, 4] <- dateofday[i]
                        adjpatr[i + (j - 1) * 365, 5] <- i %% 7 + 1
                }
        }                                       #choose links for moveables.
        mvblelinks <- (npatr + 1):(npatr + nmatr)      #choose days for moveables
        mvbledays <- sample(evendays, size = nmatr, replace = F)
        #generate MATR counts
```

```
adjmatr <- matrix(nrow = 2 * nmatr, ncol = 5)
for(i in 1:nmatr) {
        adjmatr[2 * i - 1, 2] <- rpois(n = 1, theta[mvblelinks[i]]/EF[
                mvbledays[i]])
        adjmatr[2 * i - 1, 1] <- mvblelinks[i]
        adjmatr[2 * i - 1, 3] <- monthofday[mvbledays[i]]
        adjmatr[2 * i - 1, 5] <- (mvbledays[i] %% 7) + 1
        adjmatr[2 * i - 1, 4] <- dateofday[mvbledays[i]]
        adjmatr[2 * i, 2] <- rpois(n = 1, theta[mvblelinks[i]]/EF[
                mvbledays[i] + 1])
        adjmatr[2 * i, 1] <- mvblelinks[i]
        adjmatr[2 * i, 3] <- monthofday[mvbledays[i] + 1]
        adjmatr[2 * i, 5] <- ((mvbledays[i] + 1) %% 7) + 1
        adjmatr[2 * i, 4] <- dateofday[mvbledays[i] + 1]
}                                       #choose days for sat.
firstday <- sample(c(1:7), size = 1)
satdays <- seq(firstday, by = repeatcycle %/% nsat, length = nsatdays)
#for each sat obs in each day choose an hour and set of links
sathours <- matrix(nrow = nsatobsday, ncol = nsatdays)
satlinks <- matrix(nrow = nsatobsday, ncol = nsatdays)
for(j in 1:nsatdays) {
        sathours[, j] <- sample(c(1:24), size = 1)
        for(i in 1:nsatobsday) {
                satlinks[i, j] <- sample(c(1:nlink), size = 1)
        }
}                                       #generate satobs.
adjsat <- matrix(nrow = nsatdays * nsatobsday, ncol = 5)
for(j in 1:nsatdays) {
        for(i in 1:nsatobsday) {
                linkvec <- rep(satlinks[i, j], 2 * nmatr)
                dayvec <- rep(satdays[j], 2 * nmatr)
                if(all((dayvec != mvbledays) | (linkvec != mvblelinks))
                        ) {
                        adjsat[nsatobsday * (j - 1) + i, 2] <- min(288 *
                         HEF[sathours[i, j]] * EF[satdays[j]] * rpois(
                         n = 1, theta[satlinks[i, j]]/(288 * HEF[
                         sathours[i, j]] * EF[satdays[j]])), capacity)
                        adjsat[nsatobsday * (j - 1) + i, 1] <- satlinks[
                         i, j]
                        adjsat[nsatobsday * (j - 1) + i, 3] <-
                         monthofday[satdays[j]]
                        adjsat[nsatobsday * (j - 1) + i, 4] <-
                         dateofday[satdays[j]]
                        adjsat[nsatobsday * (j - 1) + i, 5] <- (satdays[
                         j] %% 7) + 1
                }
        }
}                                       #remove missing sat rows
adjsat <- adjsat[adjsat[, 1] != "NA", ] #calc true VMT
lengths <- scan("length.out")
VMT.t <- sum(lengths[1:nlink] * theta)
#output data for traditional method
```

```
    write.table(npatr, file = "patr.out", dimnames.write = F)
    write.table(as.vector(c(1:npatr)), file = "patr.out", dimnames.write =
            F, append = T)
    write.table(adjpatr, file = "patr.out", dimnames.write = F, sep = "   ",
            append = T)
    write.table(nmatr, file = "matr.out", dimnames.write = F)
    write.table(as.vector(c((npatr + 1):(nmatr + npatr))), file =
            "matr.out", dimnames.write = F, append = T)
    write.table(adjmatr, file = "matr.out", dimnames.write = F, sep = "   ",
            append = T)
    write.table(adjsat, file = "sate.out", dimnames.write = F, sep = "   ")
    write.table(as.vector(theta), file = "aadt.out", dimnames.write = F)
    write.table(VMT.t, file = "vmt.out", dimnames.write = F)
}
```

Appendix D. Traditional Method AADT and VMT Estimation Code

```
/****************************/
/*Program of VMT Estimations*/
/* Carolyn Kan  07/21/98 */
/****************************/

#include <stdio.h
#include <string.h
#include <stdlib.h
#include <sys/types.h
#include <sys/stat.h
#include <fcntl.h

#define January 1
#define Febuary 2
#define March 3
#define April 4
#define May 5
#define June 6
#define July 7
#define August 8
#define September 9
#define October 10
#define November 11
#define December 12

#define Monday 1
#define Tuesday 2
#define Wednesday 3
#define Thursday 4
#define Friday 5
#define Saturday 6
#define Sunday 7

#define no_link 100
#define day_of_year 365
#define max_rd  36500
/* max_rd = no_link * day_of_year
suppose it is not a leap year
maximum records allowed */

/*file pointer*/
FILE *file_in;
FILE *file_out;
FILE *aadtp;

/*file names*/
char *outfile1;
char *outfile2;
char *outfile3;
char *infile1;
char *infile2;
```

```
char *infile3;
char *infile4;
char *infile5;

/*no of permenent & moving ATR and Satellite data generated */
int p_ATR; /*# of permanent ATR*/
int m_ATR; /*# of movable ATR*/
int sate; /*# of satellite images*/
int p_link[no_link]; /*list of link id for P ATR*/
int m_link[no_link]; /*list of link id for m ATR*/
int sate_link[no_link]; /*list of link id for satellite image*/
int no_mATR_rd; /*# of records for P ATR*/
int no_pATR_rd; /*# of records for m ATR*/
int no_sate_rd; /*# of records for satellite image*/

int sate_not_ATR; /*# of links without ground data only with satellite data*/
int sate_only[no_link]; /*list of link id without ground data only with
satellite data*/

struct ATR_data
{
 int linkID; /*link identification */
 float ADT;

 /*ADT value for simulated ATR and satellite data */
 int month;
 int day;
 int week;
 }; /*end of struct*/

struct sat_data
{
 int linkID; /*link identification */
 float flow;
 /*ADT value for simulated ATR and satellite data */
 int month;
 int day;
 int week;
/* float start_time; */
/* float end_time; */
 }; /*end of struct*/


struct ATR_data p_ADT[max_rd];
struct ATR_data m_ADT[max_rd];
struct sat_data sat_vol[max_rd];

float Jan_sum[no_link];
float Feb_sum[no_link];
float Mar_sum[no_link];
float Apr_sum[no_link];
float May_sum[no_link];
float Jun_sum[no_link];
float Jul_sum[no_link];
float Aug_sum[no_link];
float Sep_sum[no_link];
```

```
float Oct_sum[no_link];
float Nov_sum[no_link];
float Dec_sum[no_link];

float Mon_sum[no_link];
float Tue_sum[no_link];
float Wed_sum[no_link];
float Thu_sum[no_link];
float Fri_sum[no_link];
float Sat_sum[no_link];
float Sun_sum[no_link];

float Jan_AADT[no_link];
float Feb_AADT[no_link];
float Mar_AADT[no_link];
float Apr_AADT[no_link];
float May_AADT[no_link];
float Jun_AADT[no_link];
float Jul_AADT[no_link];
float Aug_AADT[no_link];
float Sep_AADT[no_link];
float Oct_AADT[no_link];
float Nov_AADT[no_link];
float Dec_AADT[no_link];

float Mon_AADT[no_link];
float Tue_AADT[no_link];
float Wed_AADT[no_link];
float Thu_AADT[no_link];
float Fri_AADT[no_link];
float Sat_AADT[no_link];
float Sun_AADT[no_link];

float yr_AADT[no_link];
/*365-day avg AADT */
float wk_AADT[no_link];
/*week avg AADT */
float checking[no_link];

/* declare the monthly and daily factors for each link */
float MEF_Jan[no_link];
float MEF_Feb[no_link];
float MEF_Mar[no_link];
float MEF_Apr[no_link];
float MEF_May[no_link];
float MEF_Jun[no_link];
float MEF_Jul[no_link];
float MEF_Aug[no_link];
float MEF_Sep[no_link];
float MEF_Oct[no_link];
float MEF_Nov[no_link];
float MEF_Dec[no_link];

float DEF_Mon[no_link];
float DEF_Tue[no_link];
float DEF_Wed[no_link];
```

```
float DEF_Thu[no_link];
float DEF_Fri[no_link];
float DEF_Sat[no_link];
float DEF_Sun[no_link];

/* 1/factors for calculating harmonic mean of factors*/
float tm1; float tm2;
float tm3; float tm4;
float tm5; float tm6;
float tm7; float tm8;
float tm9; float tm10;
float tm11; float tm12;
float tw1; float tw2;
float tw3; float tw4;
float tw5; float tw6;
float tw7;

/* declare the final averaged monthly and daily factors */
float MEF1; float MEF2; float MEF3;
float MEF4; float MEF5; float MEF6;
float MEF7; float MEF8; float MEF9;
float MEF10; float MEF11;
float MEF12; float DEF1;
float DEF2; float DEF3;
float DEF4; float DEF5;
float DEF6; float DEF7;

float est_AADT[no_link][5];/*declare output array [link id][true
AADT][flag][est AADT Ground only][est AADT ground+satellite]*/
/* Definitions of Flag */
/* 0 -- link without data */
/* 1 -- link with permanent ATR only */
/* 2 -- link with portable ATR only */
/* 3 -- link with satellite data only */
/* 4 -- link with permanent ATR & Satellite */
/* 5 -- link with portable ATR & Satellite */
/* 6 -- link with permanent & portable ATR */
/* 7 -- link with permanent, portable ATR & satellite */

float true[no_link]; /*temp. storage for true AADT*/
float link_vmt[no_link][5]; /*declare array for link length  [link
id][length][vmt-ground only][vmt-ground+satellite][true vmt]*/
double total_t_vmt; /* the true VMT */
double total_G_vmt; /* estimated VMT -- ground only */
double total_GS_vmt; /* estimated VMT  -- ground + satellite */
double vmt_G_err; /* Absolute value of the % error of estimated VMT --
ground only */
double vmt_GS_err; /* Absolute value of the % error of estimated VMT --
ground + satellite */

int index; /* counter for number of records read in*/

int count;
int link_order;
int p_rds;
int m_rds;
```

```
int s_rds;
int all_link;
int temp_count; /* all are counters in "for" loop*/

int d1;
int d2;
int d3;
int d4; /*temporal storage for struct */
float f1;
float f2;
float f3; /*temporal storage for struct*/

int l_id;
int p_id;
int m_id; /* temp storage for link ID*/

int mm;
int wk;

float mon_factor;
float week_factor;
float est;

/* estimate AADT for each data and save it into a array for further
calculation.*/
float mATR_est[50000][2]; /* array [link ID][est. for mATR data] */
float sat_est[50000][2]; /* array [link ID][est. for sat. data] */
float avg_ADT[50000][5]; /* [link ID][# of mATR est.][sum of est. for
mATR][# of sat est.][sum of est. for sat] */
float avg; float sat_avg; float mATR_avg;

/* Adding up the total monthly volumes and total daily volumes */
int no_Mon;
int no_Tue;
int no_Wed;
int no_Thu;
int no_Fri;
int no_Sat;
int no_Sun;

int ground_also; /*flag for checking if satellite covers the ground data also*/
int diff; /* # of links without any data= # of links - # of P ATR - # of M
ATR - # of satellite*/

/*storage for averaging est. AADT for no-data link*/
float temp_total;
float est_G_mean; /* avg aadt for ground only */
float est_GS_mean; /* avg aadt for ground + sat */
float est_G_err;
float est_GS_err;
float temp_G_err;
float temp_GS_err;

int low_G_aadt;
int low_GS_aadt;
int low_G_vmt;
```

```
int low_GS_vmt; /* all counter to count if estimation < true value */
void main()
{  /*start of main*/
int fd;

outfile1 = "result.out";
outfile2 = "adt_err.out";
outfile3 = "vmt_err.out";
infile1 = "patr.out";
infile2 = "matr.out";
infile3 = "sate.out";
infile4 = "length.out";
infile5 = "aadt.out";

aadtp = fopen("AADTest.out", "w");
file_out = fopen(outfile1, "w");
if (file_out == NULL)
     {
     printf("Cannot open output file %s \n",outfile1);
     fprintf (stderr,"Cannot open output file %s \n", outfile1);
     } /*open file for output*/

for ( link_order=0; link_order<no_link; link_order++)
  {
  est_AADT[link_order][0]=0; /* linkID*/
  est_AADT[link_order][1]=-1;  /*true AADT*/
  est_AADT[link_order][2]=0;/* flag*/
  est_AADT[link_order][3]=-1;/* estimations Ground only*/
  est_AADT[link_order][4]=-1;/* estimations ground + Satellite*/
  } /*initialize the output array */

fd = open(infile1, O_RDONLY);
file_in = fdopen(fd, "r");
/* read in simulated data for links with permenent ATR */
if (file_in == NULL)
  {
  printf("Cannot open pATR file %s \n",infile1);
  fprintf (stderr,"Cannot open pATR file %s \n", infile1);
  } /*end if */
  else
   {
   index =0; /*index of records*/
   d1=d2=d3=d4=0;
   f1=f2=f3=0.0;
   fscanf (file_in, "%d\n", &p_ATR); /*read in number of permenent ATR */
   printf("# of permenent ATR is %d \n", p_ATR);
   /* fprintf(file_out, "# of permenent ATR is %d \n", p_ATR);*/
   for ( count = 0; count < p_ATR; count++) /*read in link IDs for p_ATR */
     {
     fscanf (file_in, "%d\n", &p_link[count]);
    /* printf("link %d\n", p_link[count]);*/
     } /*end for */

   while (1)
     {
     int eof = fscanf (file_in, "%d %f %d %d %d\n", &d1,&f1,&d2,&d3,&d4);
```

```
          if (eof == EOF) break;
       p_ADT[index].linkID = d1;
       p_ADT[index].ADT = f1;
       p_ADT[index].month = d2;
       p_ADT[index].day = d3;
       p_ADT[index].week = d4;
       index=index+1;
      } /*end while */
    fclose(file_in);
    no_pATR_rd = index;
   } /* end else for reading p_ATR */
printf("\n# of P ATR records = %d\n", no_pATR_rd);
/*fprintf(file_out, "\n# of P ATR records = %d\n", no_pATR_rd);  */

/* Adding up the total monthly volumes and total daily volumes */
no_Mon = no_Tue = no_Wed = no_Thu = no_Fri = no_Sat = no_Sun = 0;

for ( p_rds=0; p_rds<no_pATR_rd; p_rds++)
  {
     l_id = p_ADT[p_rds].linkID;
     mm = p_ADT[p_rds].month;
     wk = p_ADT[p_rds].week;
/*        printf("linkID, month, week = %d, %d, %d\n", l_id, mm, wk);   */
     switch (mm)
      {
     case January:
        Jan_sum[l_id] = Jan_sum[l_id] + p_ADT[p_rds].ADT;
        break;
     case Febuary:
        Feb_sum[l_id] = Feb_sum[l_id] + p_ADT[p_rds].ADT;
        break;
     case March:
        Mar_sum[l_id] = Mar_sum[l_id] + p_ADT[p_rds].ADT;
        break;
     case April:
        Apr_sum[l_id] = Apr_sum[l_id] + p_ADT[p_rds].ADT;
        break;
     case May:
        May_sum[l_id] = May_sum[l_id] + p_ADT[p_rds].ADT;
        break;
     case June:
        Jun_sum[l_id] = Jun_sum[l_id] + p_ADT[p_rds].ADT;
        break;
     case July:
        Jul_sum[l_id] = Jul_sum[l_id] + p_ADT[p_rds].ADT;
        break;
     case August:
        Aug_sum[l_id] = Aug_sum[l_id] + p_ADT[p_rds].ADT;
        break;
     case September:
        Sep_sum[l_id] = Sep_sum[l_id] + p_ADT[p_rds].ADT;
        break;
     case October:
        Oct_sum[l_id] = Oct_sum[l_id] + p_ADT[p_rds].ADT;
        break;
     case November:
```

```
        Nov_sum[l_id] = Nov_sum[l_id] + p_ADT[p_rds].ADT;
        break;
    case December:
        Dec_sum[l_id] = Dec_sum[l_id] + p_ADT[p_rds].ADT;
        break;
    default:
        printf("%d this is not a month?!\n", mm);
    } /*end of switch (mm) */
    switch (wk)
     {
    case Monday:
        Mon_sum[l_id] = Mon_sum[l_id] + p_ADT[p_rds].ADT;
        no_Mon = no_Mon + 1;
        break;
    case Tuesday:
        Tue_sum[l_id] = Tue_sum[l_id] + p_ADT[p_rds].ADT;
        no_Tue = no_Tue + 1;
        break;
    case Wednesday:
        Wed_sum[l_id] = Wed_sum[l_id] + p_ADT[p_rds].ADT;
        no_Wed = no_Wed + 1;
        break;
    case Thursday:
        Thu_sum[l_id] = Thu_sum[l_id] + p_ADT[p_rds].ADT;
        no_Thu = no_Thu + 1;
        break;
    case Friday:
        Fri_sum[l_id] = Fri_sum[l_id] + p_ADT[p_rds].ADT;
        no_Fri = no_Fri + 1;
        break;
    case Saturday:
        Sat_sum[l_id] = Sat_sum[l_id] + p_ADT[p_rds].ADT;
        no_Sat = no_Sat + 1;
        break;
    case Sunday:
        Sun_sum[l_id] = Sun_sum[l_id] + p_ADT[p_rds].ADT;
        no_Sun = no_Sun + 1;
        break;
    default:
        printf("%d this is not a day of the week?!\n",wk);
    } /*end of switch (wk)*/
  } /* end of for(p_rds)*/

/* Calculation of expansion factors */
for (link_order=0; link_order<p_ATR; link_order++)
 {
 l_id = p_link[link_order];
 Jan_AADT[l_id]= Jan_sum[l_id]/31;
 Feb_AADT[l_id]= Feb_sum[l_id]/28;
 Mar_AADT[l_id]= Mar_sum[l_id]/31;
 Apr_AADT[l_id]= Apr_sum[l_id]/30;
 May_AADT[l_id]= May_sum[l_id]/31;
 Jun_AADT[l_id]= Jun_sum[l_id]/30;
 Jul_AADT[l_id]= Jul_sum[l_id]/31;
 Aug_AADT[l_id]= Aug_sum[l_id]/31;
 Sep_AADT[l_id]= Sep_sum[l_id]/30;
```

```
 Oct_AADT[l_id]= Oct_sum[l_id]/31;
 Nov_AADT[l_id]= Nov_sum[l_id]/30;
 Dec_AADT[l_id]= Dec_sum[l_id]/31;

 yr_AADT[l_id]=
(Jan_sum[l_id]+Feb_sum[l_id]+Mar_sum[l_id]+Apr_sum[l_id]+May_sum[l_id]+Jun_s
um[l_id]+Jul_sum[l_id]+Aug_sum[l_id]+Sep_sum[l_id]+Oct_sum[l_id]+Nov_sum[l_i
d]+Dec_sum[l_id])/365;

 /* putting the true AADT into the output array for permenent ATR */
       est_AADT[link_order][0] = l_id;
       est_AADT[link_order][3] = yr_AADT[l_id];
       est_AADT[link_order][4] = yr_AADT[l_id];
       est_AADT[link_order][2] = 1;
    /*   printf("\nFor P ATR link %f, the type is %f, the estimate is
%f\n", est_AADT[link_order][0], est_AADT[link_order][2],
est_AADT[link_order][1]); */

 /* checking the calculation */
 checking[l_id]=
(Mon_sum[l_id]+Tue_sum[l_id]+Wed_sum[l_id]+Thu_sum[l_id]+Fri_sum[l_id]+Sat_s
um[l_id]+Sun_sum[l_id])/365;
 if (yr_AADT[l_id] != checking[l_id])
   {
    fprintf(stderr,"the calclation might be wrong?!\n ");
    printf("the calclation might be wrong?!\n ");
   }
   /*end if*/

 Mon_AADT[l_id]= Mon_sum[l_id]/no_Mon;
 Tue_AADT[l_id]= Tue_sum[l_id]/no_Tue;
 Wed_AADT[l_id]= Wed_sum[l_id]/no_Wed;
 Thu_AADT[l_id]= Thu_sum[l_id]/no_Thu;
 Fri_AADT[l_id]= Fri_sum[l_id]/no_Fri;
 Sat_AADT[l_id]= Sat_sum[l_id]/no_Sat;
 Sun_AADT[l_id]= Sun_sum[l_id]/no_Sun;

 wk_AADT[l_id] =
(Mon_AADT[l_id]+Tue_AADT[l_id]+Wed_AADT[l_id]+Thu_AADT[l_id]+Fri_AADT[l_id]+
Sat_AADT[l_id]+Sun_AADT[l_id])/7;

 MEF_Jan[l_id] =yr_AADT[l_id]/Jan_AADT[l_id];
 MEF_Feb[l_id] =yr_AADT[l_id]/Feb_AADT[l_id];
 MEF_Mar[l_id] =yr_AADT[l_id]/Mar_AADT[l_id];
 MEF_Apr[l_id] =yr_AADT[l_id]/Apr_AADT[l_id];
 MEF_May[l_id] =yr_AADT[l_id]/May_AADT[l_id];
 MEF_Jun[l_id] =yr_AADT[l_id]/Jun_AADT[l_id];
 MEF_Jul[l_id] =yr_AADT[l_id]/Jul_AADT[l_id];
 MEF_Aug[l_id] =yr_AADT[l_id]/Aug_AADT[l_id];
 MEF_Sep[l_id] =yr_AADT[l_id]/Sep_AADT[l_id];
 MEF_Oct[l_id] =yr_AADT[l_id]/Oct_AADT[l_id];
 MEF_Nov[l_id] =yr_AADT[l_id]/Nov_AADT[l_id];
 MEF_Dec[l_id] =yr_AADT[l_id]/Dec_AADT[l_id];

 DEF_Mon[l_id] =wk_AADT[l_id]/Mon_AADT[l_id];
 DEF_Tue[l_id] =wk_AADT[l_id]/Tue_AADT[l_id];
```

```
 DEF_Wed[l_id] =wk_AADT[l_id]/Wed_AADT[l_id];
 DEF_Thu[l_id] =wk_AADT[l_id]/Thu_AADT[l_id];
 DEF_Fri[l_id] =wk_AADT[l_id]/Fri_AADT[l_id];
 DEF_Sat[l_id] =wk_AADT[l_id]/Sat_AADT[l_id];
 DEF_Sun[l_id] =wk_AADT[l_id]/Sun_AADT[l_id];
 } /*end for  (link_order) */

/* Averaging the MEF's and DEF's for this group of links */
/* TAKING          ONIC MEAN */
MEF1= MEF2= MEF3= MEF4= MEF5= MEF6= MEF7= MEF8= MEF9= MEF10= MEF11= MEF12=0;
DEF1= DEF2= DEF3= DEF4= DEF5= DEF6= DEF7=0;

tm1= tm2= tm3= tm4= tm5= tm6= tm7= tm8= tm9= tm10= tm11= tm12=0;
tw1= tw2= tw3= tw4= tw5= tw6= tw7=0;

for (link_order =0; link_order<p_ATR; link_order++)
  {
  l_id = p_link[link_order];
  tm1 = tm1+ (1/MEF_Jan[l_id]);
  tm2 = tm2+ (1/MEF_Feb[l_id]);
  tm3 = tm3+ (1/MEF_Mar[l_id]);
  tm4 = tm4+ (1/MEF_Apr[l_id]);
  tm5 = tm5+ (1/MEF_May[l_id]);
  tm6 = tm6+ (1/MEF_Jun[l_id]);
  tm7 = tm7+ (1/MEF_Jul[l_id]);
  tm8 = tm8+ (1/MEF_Aug[l_id]);
  tm9 = tm9+ (1/MEF_Sep[l_id]);
  tm10 = tm10+ (1/MEF_Oct[l_id]);
  tm11 = tm11+ (1/MEF_Nov[l_id]);
  tm12 = tm12+ (1/MEF_Dec[l_id]);

  tw1 = tw1+ (1/DEF_Mon[l_id]);
  tw2 = tw2+ (1/DEF_Tue[l_id]);
  tw3 = tw3+ (1/DEF_Wed[l_id]);
  tw4 = tw4+ (1/DEF_Thu[l_id]);
  tw5 = tw5+ (1/DEF_Fri[l_id]);
  tw6 = tw6+ (1/DEF_Sat[l_id]);
  tw7 = tw7+ (1/DEF_Sun[l_id]);

  } /* end of for*/

 tm1 = tm1/p_ATR;
 tm2 = tm2/p_ATR;
 tm3 = tm3/p_ATR;
 tm4 = tm4/p_ATR;
 tm5 = tm5/p_ATR;
 tm6 = tm6/p_ATR;
 tm7 = tm7/p_ATR;
 tm8 = tm8/p_ATR;
 tm9 = tm9/p_ATR;
 tm10 = tm10/p_ATR;
 tm11 = tm11/p_ATR;
 tm12 = tm12/p_ATR;

 tw1 = tw1/p_ATR;
 tw2 = tw2/p_ATR;
```

```
tw3 = tw3/p_ATR;
tw4 = tw4/p_ATR;
tw5 = tw5/p_ATR;
tw6 = tw6/p_ATR;
tw7 = tw7/p_ATR;

MEF1 = 1/tm1;
MEF2 = 1/tm2;
MEF3 = 1/tm3;
MEF4 = 1/tm4;
MEF5 = 1/tm5;
MEF6 = 1/tm6;
MEF7 = 1/tm7;
MEF8 = 1/tm8;
MEF9 = 1/tm9;
MEF10 = 1/tm10;
MEF11 = 1/tm11;
MEF12 = 1/tm12;

DEF1 = 1/tw1;
DEF2 = 1/tw2;
DEF3 = 1/tw3;
DEF4 = 1/tw4;
DEF5 = 1/tw5;
DEF6 = 1/tw6;
DEF7 = 1/tw7;

/* MEF's and DEF's are ready! */
/*printf("MEF1 = %f\n", MEF1);
printf("MEF2 = %f\n", MEF2);
printf("MEF3 = %f\n", MEF3);
printf("MEF4 = %f\n", MEF4);
printf("MEF5 = %f\n", MEF5);
printf("MEF6 = %f\n", MEF6);
printf("MEF7 = %f\n", MEF7);
printf("MEF8 = %f\n", MEF8);
printf("MEF9 = %f\n", MEF9);
printf("MEF10 = %f\n", MEF10);
printf("MEF11 = %f\n", MEF11);
printf("MEF12 = %f\n", MEF12);
printf("DEF1 = %f\n", DEF1);
printf("DEF2 = %f\n", DEF2);
printf("DEF3 = %f\n", DEF3);
printf("DEF4 = %f\n", DEF4);
printf("DEF5 = %f\n", DEF5);
printf("DEF6 = %f\n", DEF6);
printf("DEF7 = %f\n", DEF7);*/

/*fprintf(file_out, "MEF1 = %f\n", MEF1);
fprintf(file_out, "MEF2 = %f\n", MEF2);
fprintf(file_out, "MEF3 = %f\n", MEF3);
fprintf(file_out, "MEF4 = %f\n", MEF4);
fprintf(file_out, "MEF5 = %f\n", MEF5);
fprintf(file_out, "MEF6 = %f\n", MEF6);
fprintf(file_out, "MEF7 = %f\n", MEF7);
fprintf(file_out, "MEF8 = %f\n", MEF8);
```

```
fprintf(file_out, "MEF9 = %f\n", MEF9);
fprintf(file_out, "MEF10 = %f\n", MEF10);
fprintf(file_out, "MEF11 = %f\n", MEF11);
fprintf(file_out, "MEF12 = %f\n", MEF12);
fprintf(file_out,"DEF1 = %f\n", DEF1);
fprintf(file_out,"DEF2 = %f\n", DEF2);
fprintf(file_out,"DEF3 = %f\n", DEF3);
fprintf(file_out,"DEF4 = %f\n", DEF4);
fprintf(file_out,"DEF5 = %f\n", DEF5);
fprintf(file_out,"DEF6 = %f\n", DEF6);
fprintf(file_out,"DEF7 = %f\n", DEF7);*/


fd = open(infile2, O_RDONLY);
/*printf("open %s as fd %d\n",infile2, fd); */
file_in = fdopen(fd, "r");
/* read in simulated data for links with portable ATR */
if (file_in == NULL)
 {
  printf("Cannot open mATR file %s\n", infile2);
  fprintf (stderr,"Cannot open mATR file %s\n", infile2);
 } /*end if */
 else
  {
   index = 0;
   d1=d2=d3=d4=0;
   f1=f2=f3=0.0;
   fscanf (file_in, "%d", &m_ATR); /*read in number of portable ATR*/
   printf("# of movable ATR is %d \n", m_ATR);
   /* fprintf(file_out, "# of movable ATR is %d \n", m_ATR);*/
   for ( count = 0; count < m_ATR; count++) /*read in link IDs for m_ATR*/
     {
      fscanf (file_in, "%d", &m_link[count]);
    /*  printf("link %d \n", m_link[count]); */
     } /*end for */

   while (1)
     {
      int eof = fscanf (file_in, "%d %f %d %d %d",&d1,&f1,&d2,&d3,&d4);
      if (eof == EOF) break;
      m_ADT[index].linkID = d1;
      m_ADT[index].ADT = f1;
      m_ADT[index].month = d2;
      m_ADT[index].day = d3;
      m_ADT[index].week = d4;
      index=index+1;
     } /*end while */
   } /* end else for reading m_ATR */
 fclose(file_in);
 no_mATR_rd = index;
 printf("# of mATR records = %d\n", no_mATR_rd);
 /*fprintf(file_out, "# of mATR records = %d\n", no_mATR_rd); */

/* save m_ATR flag into output array*/
for (count=0; count < m_ATR; count++)
  {
```

```
   for (link_order=0; link_order < no_link; link_order ++)
    {
     if (est_AADT[link_order][0] == m_link[count] )
      {
       est_AADT[link_order][2]=6;
    /*   printf("\nfor MATR link %f, it is also PATR. The type is %f\n",
est_AADT[link_order][0], est_AADT[link_order][2]); */
       break;
      } else if (est_AADT[link_order][0]==0)
         {
          est_AADT[link_order][0]= m_link[count];
          est_AADT[link_order][2] = 2;
          break;
         } /* end else if*/
    } /*end link_order*/
   } /*end count*/

/* estimate the AADT for links with movable ATR */
for ( m_rds=0; m_rds<no_mATR_rd; m_rds++)
  {
   l_id = m_ADT[m_rds].linkID;
   mm = m_ADT[m_rds].month;
   wk = m_ADT[m_rds].week;
 /*  printf("lid = %d, month = %d, week = %d\n",l_id,mm,wk);      */
   switch (mm)
      {
     case January:
         mon_factor = MEF1;
         break;
     case Febuary:
         mon_factor = MEF2;
         break;
     case March:
         mon_factor = MEF3;
         break;
     case April:
         mon_factor = MEF4;
         break;
     case May:
         mon_factor = MEF5;
         break;
     case June:
         mon_factor = MEF6;
         break;
     case July:
         mon_factor = MEF7;
         break;
     case August:
         mon_factor = MEF8;
         break;
     case September:
         mon_factor = MEF9;
         break;
     case October:
         mon_factor = MEF10;
         break;
```

```
    case November:
        mon_factor = MEF11;
        break;
    case December:
        mon_factor = MEF12;
        break;
    default:
        printf("%d this is not a month?!\n",mm);
    } /*end of switch (mm) */
    switch (wk)
     {
    case Monday:
        week_factor = DEF1;
        break;
    case Tuesday:
        week_factor = DEF2;
        break;
    case Wednesday:
        week_factor = DEF3;
        break;
    case Thursday:
        week_factor = DEF4;
        break;
    case Friday:
        week_factor = DEF5;
        break;
    case Saturday:
        week_factor = DEF6;
        break;
    case Sunday:
        week_factor = DEF7;
        break;
    default:
        printf("%d this is not a day of the week?!\n", wk);
    } /*end of switch (wk) */
  /* printf("mon_factor = %f, week_factor = %f, data = %f\n", mon_factor,
week_factor, m_ADT[m_rds].ADT); */
      /*temporal storage for eatimeated AADT for one daily data */
   est = m_ADT[m_rds].ADT * mon_factor * week_factor;
   mATR_est[m_rds][1] = est;
   mATR_est[m_rds][0] = l_id;
  /* printf("for link %f, estimated aadt is %f\n",mATR_est[m_rds][0],
mATR_est[m_rds][1]);   */

  } /* end of for(m_rds)*/


/* Calculate ground-only Average AADT for each link.*/
/*initialize the array of estimates*/
for (link_order=0; link_order<m_ATR; link_order++)
  {
   avg_ADT[link_order][0]= 0; /*link id */
   avg_ADT[link_order][1]= 0; /*# of mATR estimates*/
   avg_ADT[link_order][2]= 0; /*sum of mATR estimates*/
   avg_ADT[link_order][3]= 0; /*# of sat estimates*/
   avg_ADT[link_order][4]= 0; /*sum of sat estimates*/
```

```
  }

 /* for movable ATR */
 for (count =0; count < no_mATR_rd; count ++)
   {
   for (link_order=0; link_order<m_ATR; link_order++)
     {
     if (m_link[link_order]== mATR_est[count][0])
       {
       avg_ADT[link_order][0] = mATR_est[count][0];  /*link id*/
       avg_ADT[link_order][1]= avg_ADT[link_order][1]+1;  /*# of estimates */
       avg_ADT[link_order][2] = avg_ADT[link_order][2] +
mATR_est[count][1]; /* sum of estimates */
       } /*end if*/
     } /*end for link_order */
   } /*end for count  */

   /* testing
   for (link_order=0; link_order<no_link; link_order++)
     {
     printf("\nlink ID = %f, # of est. = %f, sum of est. = %f",
avg_ADT[link_order][0],avg_ADT[link_order][1],avg_ADT[link_order][2]);
     }  end for*/

 /* averaging ground-only estimates of AADT */
 for (link_order=0; link_order<no_link; link_order++)
   {
   if (avg_ADT[link_order][0]==0)
     {
     break;
     } else
       {
       for (count = 0; count< no_link; count ++)
          {
          mATR_avg =  avg_ADT[link_order][2]/avg_ADT[link_order][1];
          if (est_AADT[count][0] == avg_ADT[link_order][0])   /* link
ID match*/
           {
           est_AADT[count][3] = mATR_avg;
           } /*end if*/
          } /* end for count*/
       } /* end else*/
   } /*end for link_order*/

 /* check if there are links without ground-only data */
 if (m_ATR + p_ATR  no_link)
   {
   printf("ALERT! SOMETHING IS WRONG! m_ATR + p_ATR  no_link\n");
   printf("m_ATR = %d ,  p_ATR = %d \n", m_ATR, p_ATR);
   fprintf(file_out, "ALERT! SOMETHING IS WRONG! m_ATR + p_ATR  no_link\n");
   fprintf(file_out, "m_ATR = %d ,  p_ATR = %d\n", m_ATR, p_ATR);
   } /* end if */
  else if (m_ATR + p_ATR < no_link)
       {
       diff = no_link - m_ATR - p_ATR;
       printf("There are %d links without any ground data.\n", diff);
```

```
        /* fprintf(file_out, "There are %d links without any ground data.\n",
diff);*/
      } /*end else if*/

/* Use ARITHMETIC MEAN of estimated AADT as the estimations for the links
without ground-only data */
temp_total =0;
for (count=0; count <no_link; count++)
  {
  if (est_AADT[count][0]!=0)
    {
    temp_total = temp_total+ est_AADT[count][3];
    } /* end if*/
  } /* end for*/
est_G_mean = temp_total/(no_link - diff);


/* Read in Length of the links*/
fd = open(infile4, O_RDONLY);
file_in = fdopen(fd, "r");
if (file_in == NULL)
  {
  printf("Cannot open length file %s \n",infile4);
  fprintf (stderr,"Cannot open length file %s \n", infile4);
  } /*end if */
 else
   {
   index =0; /*index of records*/
   f1=f2=f3=0.0;
   for (count=0; count <no_link; count ++)
     {
     fscanf (file_in, "%f %f\n", &f1, &f2);
     link_vmt[count][0] = f1;  /* link Id*/
     link_vmt[count][1] = f2;  /* link length*/
     link_vmt[count][2] = 0; /* initialize vmt ground only*/
     link_vmt[count][3] = 0; /* initialize vmt ground + satellite*/
     index=index+1;
     } /*end for count*/
   fclose(file_in);
   } /* end else for reading length */

if (index != no_link) /* checking if having exact # of length*/
  {
  printf("\n# of link length is not equal to # of links?!");
  fprintf(file_out,"\n# of link length is not equal to # of links?!");
  } /* end if*/



/* Calculate ground-only VMT */

for (count =0; count <no_link; count ++)
  {
  for (link_order =0; link_order< no_link; link_order++)
    {
    if (link_vmt[count][0] == est_AADT[link_order][0])
```

```
      {
      link_vmt[count][2] = link_vmt[count][1] * est_AADT[link_order][3];
      break;
      } /* end if*/
    } /* end for link_order*/
   }/* end for count*/

index =0;
for (count =0; count <no_link; count ++)
  {
  if (est_AADT[count][2]==0) index = index+1;
  } /*end for count*/
/* check if the # of links without ground-only data is correct*/
if (index != diff)
  {
  printf("\nproblem about # of links without data?!");

  fprintf(file_out, "\nproblem about # of links without data?!");
  } else
    {
    for (count=0; count <no_link; count ++)
      {
      if (link_vmt[count][2] ==0)
        {
        link_vmt[count][2] = link_vmt[count][1] * est_G_mean; /* use
average est. AADT for links without data*/
        } /* end if*/
      }/* end for count*/
    } /*end else*/

total_G_vmt =0;
for (count =0; count <no_link; count ++)
  {
  if (link_vmt[count][2]==0)
    {
    printf("We got problem for vmt array?!");
    fprintf(file_out, "We got problem for vmt array?!");
    break;
    } else
      {
      total_G_vmt = total_G_vmt + link_vmt[count][2];
      } /*end else*/
  } /* end for*/

printf("\nThe total ground-only VMT for %d links in this class is %f.\n" ,
no_link, total_G_vmt);
/*fprintf(file_out, "%f   " ,total_G_vmt); */
 /* end printing ground-only VMT */


fd = open(infile3, O_RDONLY);
/*printf("\n open %s as fd %d\n", infile3, fd);*/
file_in = fdopen(fd,"r");
/* read in simulated data for links having satellite data */
if (file_in == NULL)
  {
```

```
  printf("\n Cannot open satellite file %s", infile3);
  fprintf (stderr,"\n Cannot open satellite file %s", infile3);
 } /*end if*/
 else
  {
   index = 0;
   d1=d2=d3=d4=0;
   f1=f2=f3=0.0;
   while (1)
   /* there is no input for number of satellite data and no link
information */
     {
      int eof = fscanf (file_in, "%d %f %d %d %d",&d1,&f1,&d2,&d3,&d4);
      if ( eof == EOF) break;
      sat_vol[index].linkID = d1;
      sat_vol[index].flow = f1; /*the input is 24-hr volume from Roger's
program*/
      sat_vol[index].month = d2;
      sat_vol[index].day = d3;
      sat_vol[index].week = d4;
     /* sat_vol[index].start_time = f2;  */
     /* sat_vol[index].end_time = f3;    */
      index=index+1;
     } /*end while*/
   } /* end else for reading sate */
no_sate_rd = index;
printf("\n# of sate records is %d\n", no_sate_rd);
/*fprintf(file_out, "\n# of sate records is %d\n", no_sate_rd);*/

/* search for the # of links and list of link id*/
for (link_order =0; link_order <no_link; link_order++)
  {
  sate_link[link_order]=0;
  } /*initialize the array*/
sate =0;


/*sort the list of link ID*/
for (count=0; count< no_sate_rd; count++)
  {
  for (link_order =0; link_order<no_link; link_order++)
    {
    if (sat_vol[count].linkID == sate_link[link_order])
      {
       break;
      } else if (sate_link[link_order]==0)
         {
          sate_link[link_order]= sat_vol[count].linkID;
          sate = sate +1;
          break;
         } /* end else if*/
      } /* end for link_order*/
   } /* end for count*/

fclose(file_in);
```

```
/* print # of satellite and the list of link ID*/
printf("# of satellite = %d\n", sate);
/*fprintf(file_out, "# of satellite = %d\n", sate);*/
/*for (link_order=0; link_order<sate; link_order++)
  {
  printf("the sate link is %d\n", sate_link[link_order]);
  } end for print*/

/* estimate the AADT for links with satellite data */
for ( s_rds=0; s_rds<no_sate_rd; s_rds++)
  {
  l_id = sat_vol[s_rds].linkID;
  mm = sat_vol[s_rds].month;
  wk = sat_vol[s_rds].week;
  switch (mm)
      {
    case January:
        mon_factor = MEF1;
        break;
    case Febuary:
        mon_factor = MEF2;
        break;
    case March:
        mon_factor = MEF3;
        break;
    case April:
        mon_factor = MEF4;
        break;
    case May:
        mon_factor = MEF5;
        break;
    case June:
        mon_factor = MEF6;
        break;
    case July:
        mon_factor = MEF7;
        break;
    case August:
        mon_factor = MEF8;
        break;
    case September:
        mon_factor = MEF9;
        break;
    case October:
        mon_factor = MEF10;
        break;
    case November:
        mon_factor = MEF11;
        break;
    case December:
        mon_factor = MEF12;
        break;
    default:
        printf("%d this is not a month?!\n",mm);
    } /*end of switch (mm) */
    switch (wk)
```

```
      {
    case Monday:
        week_factor = DEF1;
        break;
    case Tuesday:
        week_factor = DEF2;
        break;
    case Wednesday:
        week_factor = DEF3;
        break;
    case Thursday:
        week_factor = DEF4;
        break;
    case Friday:
        week_factor = DEF5;
        break;
    case Saturday:
        week_factor = DEF6;
        break;
    case Sunday:
        week_factor = DEF7;
        break;
    default:
        printf("%d this is not a day of the week?!\n", wk);
    } /*end of switch (wk) */
   /* printf("mon_factor = %f, week_factor = %f, data = %f\n", mon_factor,
week_factor, sat_vol[s_rds].flow);*/
       /*temporal storage for eatimeated AADT for one daily data */
    est = sat_vol[s_rds].flow * mon_factor * week_factor;
    sat_est[s_rds][1] = est;
    sat_est[s_rds][0] = l_id;
  /*  printf("\nFor link %f, estimated aadt is %f\n",sat_est[s_rds][0],
sat_est[s_rds][1]);  */

  } /* end of for(s_rds)*/



/* Calculate ground+satellite Average AADT for each link.*/
 /* for Satellite */
/* First to search for links with only satellite data*/
/* also save flags for links w/ p_ATR and Satellite and for links w/ m_ATR
and Sate*/
/* already KNOWN flag type 1,2, & 6 */
for (count=0; count < sate; count ++)
  {
   for (link_order =0; link_order<no_link; link_order++)
     {
     if
((sate_link[count]==est_AADT[link_order][0])&&(est_AADT[link_order][2]==1))
       {
        est_AADT[link_order][2]=4;
         break;
       } /* end if the link is pATR */
     if
((sate_link[count]==est_AADT[link_order][0])&&(est_AADT[link_order][2]==2))
```

```
      {
      est_AADT[link_order][2]=5;
      break;
      } /* end if the link is mATR */
     if
((sate_link[count]==est_AADT[link_order][0])&&(est_AADT[link_order][2]==6))
      {
      est_AADT[link_order][2]=7;
      break;
      } /* end if the link is pATR + mATR */
     if ((sate_link[count]==est_AADT[link_order][0])&&(
est_AADT[link_order][2]==3))
      {
      break;
      } /* end if the link is sate */
     if (est_AADT[link_order][0]==0)
      {
      est_AADT[link_order][0] = sate_link[count];
      est_AADT[link_order][2] = 3;
      break;
      } /* end if the link is new */
     } /* end for link_order*/
   } /* end for count*/


/* add satellite estimates into averaging array for sate-only links and for
links w/ m ATR and Sate */
for (count =0; count < no_sate_rd; count ++)
  {
   for (link_order=0; link_order<no_link; link_order++)
     {
     if (avg_ADT[link_order][0]== sat_est[count][0]) /* link ID match*/
      {
      avg_ADT[link_order][3]= avg_ADT[link_order][3]+1;  /*# of estimates */
      avg_ADT[link_order][4] = avg_ADT[link_order][4] +
sat_est[count][1]; /* sum of estimates */
      break;
      } else if (avg_ADT[link_order][0]==0 )
         {
         avg_ADT[link_order][3]= avg_ADT[link_order][3]+1;  /*# of
estimates */
         avg_ADT[link_order][4] = avg_ADT[link_order][4] +
sat_est[count][1]; /* sum of estimates */
         avg_ADT[link_order][0] = sat_est[count][0]; /* link ID*/
         break;
         } /*end else if*/
     } /*end for link_order */
   } /*end for count  */

   /* testing
   for (link_order=0; link_order<no_link; link_order++)
     {
     printf("\nlink ID = %f, # of est. = %f, sum of est. = %f\n",
avg_ADT[link_order][0],avg_ADT[link_order][3],avg_ADT[link_order][4]);
     } end for*/
```

```
/* averaging ground+satellite estimates of AADT */
for (link_order=0; link_order<no_link; link_order++)
  {
  if (avg_ADT[link_order][0]==0)
   {
    break;
   } else
     {
     for (count = 0; count< no_link; count ++)
         {
         sat_avg =  avg_ADT[link_order][4]/avg_ADT[link_order][3];
         avg = ((avg_ADT[link_order][2] + avg_ADT[link_order][4]) /
(avg_ADT[link_order][1] + avg_ADT[link_order][3]));
         if ((est_AADT[count][0] ==
avg_ADT[link_order][0])&&(est_AADT[count][2]!=1)&&(est_AADT[count][2]!=4))
/* link ID match and it is not pATR link*/
          {
          est_AADT[count][4] = avg;
          } /*end if*/
        } /* end for count*/
     } /* end else*/
  } /*end for link_order*/

sate_not_ATR=0;
for (count =0; count <no_link; count ++)
 {
 if (est_AADT[count][2]==3)
   {
    sate_not_ATR = sate_not_ATR +1;
   } /* end if*/
 }/* end for count*/

/* check if there are links without data */
diff =0; /*initialize*/

if (m_ATR + sate_not_ATR + p_ATR  no_link)
 {
  printf("ALERT! SO             G IS WRONG! m_ATR + p_ATR + sate_not_ATR <
no_link\n");
  printf("m_ATR = %d ,  p_ATR = %d ,  sate_not_ATR = %d\n", m_ATR, p_ATR,
sate_not_ATR);
  fprintf(file_out, "ALERT! SO             G IS WRONG! m_ATR + p_ATR +
sate_not_ATR < no_link\n");
  fprintf(file_out, "m_ATR = %d ,  p_ATR = %d ,  sate_not_ATR = %d\n",
m_ATR, p_ATR, sate_not_ATR);
 } /* end if */
 else if (m_ATR + sate_not_ATR + p_ATR < no_link)
     {
     diff = no_link - m_ATR - p_ATR - sate_not_ATR;
     printf("There are %d links without any ground data or satellite
data.\n", diff);
          /* fprintf(file_out, "There are %d links without any ground data or
satellite data.\n", diff);*/
     } /*end else if*/

/* Use ARITHMETIC MEAN of estimated AADT as the estimations for the links
```

```
without ground+ sat data */
temp_total =0;
for (count=0; count <no_link; count++)
  {
  if (est_AADT[count][0]!=0)
    {
    temp_total = temp_total+ est_AADT[count][4];
    } /* end if*/
  } /* end for*/
est_GS_mean = temp_total/(no_link - diff);


/* Calculate ground+satellite VMT */
for (count =0; count <no_link; count ++)
  {
  for (link_order =0; link_order< no_link; link_order++)
    {
    if (link_vmt[count][0] == est_AADT[link_order][0])
      {
      link_vmt[count][3] = link_vmt[count][1] * est_AADT[link_order][4];
      break;
      } /* end if*/
    } /* end for link_order*/
  }/* end for count*/

index =0;
for (count =0; count <no_link; count ++)
  {
  if (est_AADT[count][2]==0) index = index+1;
  } /*end for count*/
/* check if the # of links without data is correct*/
if (index != diff)
  {
  printf("\nproblem about # of links without data?!");

  fprintf(file_out, "\nproblem about # of links without data?!");
  } else
     {
     for (count=0; count <no_link; count ++)
       {
       if (link_vmt[count][3] ==0)
         {
         link_vmt[count][3] = link_vmt[count][1] * est_GS_mean; /* use
average est. AADT for links without data*/
         } /* end if*/
       }/* end for count*/
     } /*end else*/

total_GS_vmt =0;
for (count =0; count <no_link; count ++)
  {
  if (link_vmt[count][3]==0)
    {
    printf("We got problem for vmt array?!");
    fprintf(file_out, "We got problem for vmt array?!");
    break;
```

```
    } else
       {
       total_GS_vmt = total_GS_vmt + link_vmt[count][3];
       } /*end else*/
   } /* end for*/

/* print out the estimated AADT for the links with data */
printf("\n 0 -- link without data\n");
printf("\n 1 -- link with permanent ATR only\n");
printf("\n 2 -- link with portable ATR only\n");
printf("\n 3 -- link with satellite data only\n");
printf("\n 4 -- link with permanent ATR & Satellite\n");
printf("\n 5 -- link with portable ATR & Satellite\n");
printf("\n 6 -- link with permanent & portable ATR\n");
printf("\n 7 -- link with permanent, portable ATR & satellite\n");

/*fprintf(file_out, "\n 0 -- link without data\n");
fprintf(file_out, "\n 1 -- link with permanent ATR only\n");
fprintf(file_out, "\n 2 -- link with portable ATR only\n");
fprintf(file_out, "\n 3 -- link with satellite data only\n");
fprintf(file_out, "\n 4 -- link with permanent ATR & Satellite\n");
fprintf(file_out, "\n 5 -- link with portable ATR & Satellite\n");
fprintf(file_out, "\n 6 -- link with permanent & portable ATR\n");
fprintf(file_out, "\n 7 -- link with permanent, portable ATR &
satellite\n"); */

/*for (link_order=0; link_order<no_link; link_order++)
  {
  printf("\nFor link %f, the type is %f, estimated AADT = %f\n",
est_AADT[link_order][0], est_AADT[link_order][2], est_AADT[link_order][4]);
   fprintf(file_out,"\nFor link %f, the type is %f, estimated AADT =
%f\n",est_AADT[link_order][0],est_AADT[link_order][2],
est_AADT[link_order][4]);
  } end for */

printf("\nThe total ground+satellite VMT for %d links in this class is
%f.\n" , no_link, total_GS_vmt);
/* */
/* end of printing results*/


/* testing */
for (count =0; count < no_link; count ++)
  {
  printf("\nFor link %f, the type is %f, the true AADT is %f, the G_AADT
is %f, the GS_AADT is %f.\n",
est_AADT[count][0],est_AADT[count][2],est_AADT[count][1],est_AADT[count][3],
est_AADT[count][4]);
  } /* end for count*/


/* Read True AADT for all links */
fd = open(infile5, O_RDONLY);
file_in = fdopen(fd, "r");
if (file_in == NULL)
  {
```

```
  printf("Cannot open length file %s \n",infile4);
  fprintf (stderr,"Cannot open length file %s \n", infile4);
  } /*end if */
 else
   {
   index =0;
   f1 =0.0;
   for (count=1; count <=no_link; count ++)
     {
     fscanf (file_in, "%f \n", &f1);
     true[count]=f1;
     index=index+1;
     } /*end for count*/
   fclose(file_in);
   } /* end else for reading true AADT */

/* check if # of true AADT = # of links*/
if (index != no_link)
  {
  printf("# of true AADT is not # of links!!!");
 /*  fprintf(file_out, "# of true AADT is not # of links!!!");*/
  } /* end if */

/* testing*/
printf("G-mean = %f, GS-mean = %f.\n", est_G_mean, est_GS_mean);
for (count =1; count <=no_link; count++)
  {
  printf("\nTrue AADT =%f",true[count]);
  }


/* put true AADT into array est_AADT */
for (count =1; count <= no_link; count ++)
  {
  for (link_order=0; link_order < no_link; link_order ++)
    {
    if ((est_AADT[link_order][2] ==3) &&(est_AADT[link_order][0] == count))
      {
      est_AADT[link_order][1] = true[count];
      est_AADT[link_order][3] = est_G_mean;
      break;
      }
    if ((est_AADT[link_order][0] == count) && (est_AADT[link_order][2]
!=0)&&(est_AADT[link_order][2]!=3))
      {
      est_AADT[link_order][1] = true[count];
      break;
      }
    if ((est_AADT[link_order][2] ==0) && (est_AADT[link_order][0]==count))
      {
      break;
      }
    if ((est_AADT[link_order][0]==0)&&(est_AADT[link_order][2]==0))
      {
      est_AADT[link_order][0] = count;
      est_AADT[link_order][1] = true[count];
```

```
      est_AADT[link_order][3] = est_G_mean;
      est_AADT[link_order][4] = est_GS_mean;
      break;
      }
    } /* end for link_order*/
  } /* end for count */

/* testing */
for (count =0; count < no_link; count ++)
  {
  printf("\nFor link %f, the type is %f, the true AADT is %f, the G_AADT
is %f, the GS_AADT is %f.\n",
est_AADT[count][0],est_AADT[count][2],est_AADT[count][1],est_AADT[count][3],
est_AADT[count][4]);
  } /* end for count*/

/* Calculate True VMT */
for (count =0; count <no_link; count ++)
  {
  for (link_order =0; link_order< no_link; link_order++)
    {
    if (link_vmt[count][0] == est_AADT[link_order][0])
      {
      link_vmt[count][4] = link_vmt[count][1] * est_AADT[link_order][1];
      break;
      } /* end if*/
    } /* end for link_order*/
  }/* end for count*/

total_t_vmt =0;
for (count =0; count <no_link; count ++)
  {
  total_t_vmt = total_t_vmt + link_vmt[count][4];
  } /* end for*/

fprintf(file_out, "%f  " , total_t_vmt);
fprintf(file_out, "%f  " ,total_G_vmt);
fprintf(file_out, "%f\n", total_GS_vmt);
fclose(file_out);

/* open for % error of VMT file and calculate %error of VMT*/
file_out = fopen(outfile3, "w");
if (file_out == NULL)
     {
     printf("Cannot open output file %s \n",outfile3);
     fprintf (stderr,"Cannot open output file %s \n", outfile3);
     } /*open output file for % error file*/

low_G_vmt = low_GS_vmt =0;
vmt_G_err = (total_G_vmt - total_t_vmt)/total_t_vmt;
if (vmt_G_err <0)
  {
  vmt_G_err = - vmt_G_err;
  low_G_vmt = low_G_vmt +1;
  }
vmt_GS_err = (total_GS_vmt - total_t_vmt)/total_t_vmt;
```

```
if (vmt_GS_err <0)
  {
   vmt_GS_err = - vmt_GS_err;
   low_GS_vmt = low_GS_vmt +1;
  }

fprintf(file_out, "%f   %f\n", vmt_G_err, vmt_GS_err);

fclose(file_out);

/* open for % error of AADT file*/
file_out = fopen(outfile2, "w");
if (file_out == NULL)
       {
        printf("Cannot open output file %s \n",outfile2);
        fprintf (stderr,"Cannot open output file %s \n", outfile2);
       } /*open output file for % error of AADT file*/

/* Calculate the square percent Error of estimated AADT and print them into
another file*/
est_G_err = 0.0;
est_GS_err = 0.0;
temp_G_err =0.0;
temp_GS_err =0.0;
low_G_aadt = low_GS_aadt = 0;
for (count =0; count < no_link; count ++)
  {
   temp_G_err =  (est_AADT[count][3]-est_AADT[count][1])/est_AADT[count][1];
   if (temp_G_err < 0)
     {
      low_G_aadt = low_G_aadt +1;
     }/* end if */
   est_G_err = est_G_err + (temp_G_err* temp_G_err);
   temp_GS_err = (est_AADT[count][4]-est_AADT[count][1])/est_AADT[count][1];
   if (temp_GS_err < 0)
     {
      low_GS_aadt = low_GS_aadt +1;
     }/* end if */
   est_GS_err = est_GS_err + (temp_GS_err*temp_GS_err);
  } /* end for */

fprintf(file_out, "%f   %f\n", est_G_err, est_GS_err);

fclose(file_out);

printf("\n %d out of 100 est. AADT(G) are lower than true AADT.", low_G_aadt);
printf("\n %d out of 100 est. AADT(GS) are lower than true AADT.",
low_GS_aadt);
printf("\n %d out of 100 est. VMT(G) are lower than true VMT.", low_G_vmt);
printf("\n %d out of 100 est. VMT(GS) are lower than true VMT.", low_GS_vmt);

for (count = 0; count<no_link; count++)
fprintf(aadtp, "%f   %f   %f   %f\n", est_AADT[count][0],
est_AADT[count][1], est_AADT[count][3], est_AADT[count][4]);

fclose(aadtp);
```

```
} /*end of main*/
```

Appendix E. Model-Based Estimation Code

****

THIS PROGRAM IS WRITTEN IN S-PLUS

```
function(seed, param, reps)
{
#
# This function generates the model based AADT and VMT estimates for a fixed
# set of input parameters.
#
# The output is true VMT, ground VMT estimate, ground & sat VMT estimate,
# ground residual, and ground & sat residual.
#
# 'seed' must be a negative integer
#
# 'param' is the input parameters for run.traffic
#
# The simulation is run 'reps' times (the seed is incremented each time).
#
        AADT.G <- numeric(100)
        AADT.GS <- numeric(100)
        var.G <- numeric(100)
        var.GS <- numeric(100)
        MGE <- numeric(100)
        MGSE <- numeric(100)
        nsatobs <- (param[1] * param[6] * 365)/param[5]
        print(nsatobs)
        vmt <- matrix(nrow = reps, ncol = 5)
        write.table(param, file = "input", append = F)
        for(j in 1:reps) {
                write(seed - j + 1, file = "idum.in", append = F)
                unix("cat input | run.traffic")
                input.mat <- matrix(scan("design.out"), byrow = T, ncol = 4)
                lengths <- matrix(scan("length.out"), byrow = 2, ncol = 2)[, 2]
                AADT.T <- scan("aadt.out")
                nrow <- length(input.mat[, 1])
                wts <- as.vector(c(rep(param[8]^-2, nsatobs), rep(param[10]^-2,
                        nrow - nsatobs)))
                logCounts <- log(input.mat[, 1]) + 1/(2 * wts)
                links.G <- unique(input.mat[(nsatobs + 1):nrow, 2])
                links.GS <- unique(input.mat[, 2])
                dow.factor <- as.character(input.mat[, 3])
                month.factor <- as.character(input.mat[, 4])
                link.factor <- as.character(input.mat[, 2])
                lm.GS <- lm(logCounts ~ link.factor + dow.factor + month.factor,
                        weights = wts)
                lm.G <- lm(logCounts ~ link.factor + dow.factor + month.factor,
```

```
                subset = as.vector((nsatobs + 1):nrow))
        coef.G <- dummy.coef(lm.G)$link.factor
        mean.G <- dummy.coef(lm.G)$"(Intercept)"
        coef.GS <- dummy.coef(lm.GS)$link.factor
        mean.GS <- dummy.coef(lm.GS)$"(Intercept)"
        summary.G <- summary(lm.G)
        summary.GS <- summary(lm.GS)
        sigma.G <- summary.G$sigma
        sigma.GS <- summary.GS$sigma
        par.cov.G <- (sigma.G^2) * (summary.G$cov.unscaled)
        par.cov.GS <- (sigma.GS^2) * (summary.GS$cov.unscaled)
        for(k in 1:max(links.G)) {
                c.k <- as.matrix(c(1, rep(0, k - 1), 1, rep(0, ncol(
                        par.cov.G) - k - 1)))
                var.G[k] <- crossprod(c.k, par.cov.G %*% c.k)
                AADT.G[k] <- exp(mean.G + coef.G[as.character(k)] -
                        var.G[k]/2)
        }
        for(k in (max(links.G) + 1):100) {
                var.G[k] <- par.cov.G[1, 1]
                AADT.G[k] <- exp(mean.G - var.G[k]/2)
        }
        for(k in links.GS) {
                l <- as.numeric(row.names(as.data.frame(links.GS))[
                        links.GS == k])
                c.k <- as.matrix(c(1, rep(0, l - 1), 1, rep(0, ncol(
                        par.cov.GS) - l - 1)))
                var.GS[k] <- crossprod(c.k, par.cov.GS %*% c.k)
                AADT.GS[k] <- exp(mean.GS + coef.GS[as.character(k)] -
                        var.GS[k]/2)
        }
        links <- seq(1:100)
        links.no.GS <- links[ - links.GS]
        for(k in links.no.GS) {
                var.GS[k] <- par.cov.GS[1, 1]
                AADT.GS[k] <- exp(mean.GS - var.GS[k]/2)
        }
        VMT.T <- sum(lengths * AADT.T)
        VMT.G <- sum(lengths * AADT.G)
        VMT.GS <- sum(lengths * AADT.GS)
        MGE[j] <- sqrt(mean(((AADT.T - AADT.G)/AADT.T)^2))
        MGSE[j] <- sqrt(mean(((AADT.T - AADT.GS)/AADT.T)^2))
        vmt[j, ] <- c(VMT.T, VMT.G, VMT.GS, MGE[j], MGSE[j])
    }
    vmt     #This is the real one, next line is temporary
#   cbind(MGE, MGSE)
}
```

Appendix F. Scatterplots of the Traditional Estimation Method *vs.* the Model-Based Estimation Method (100 replications; $M = ...$)

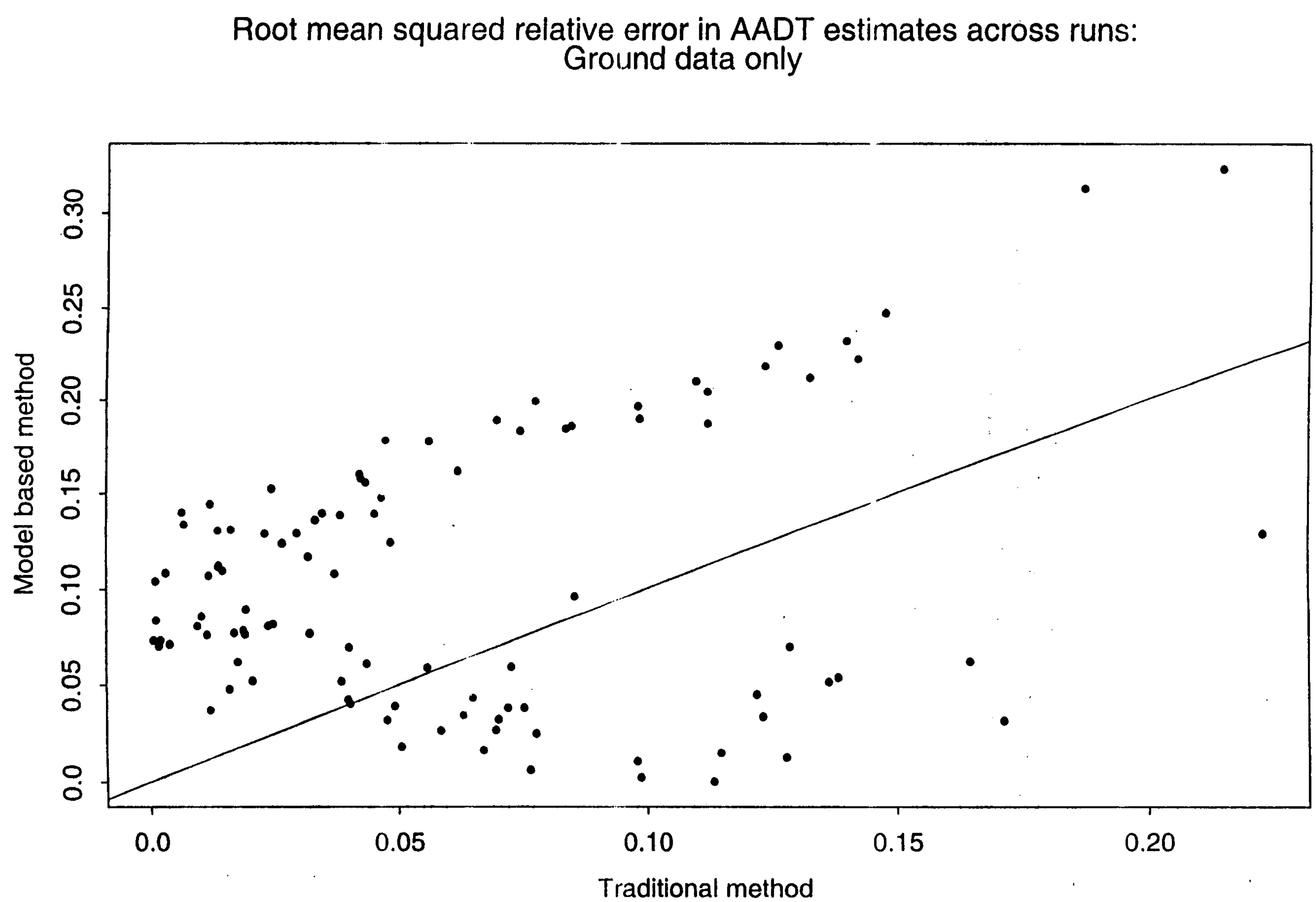


Figure F1. 25 moveable ATRs.

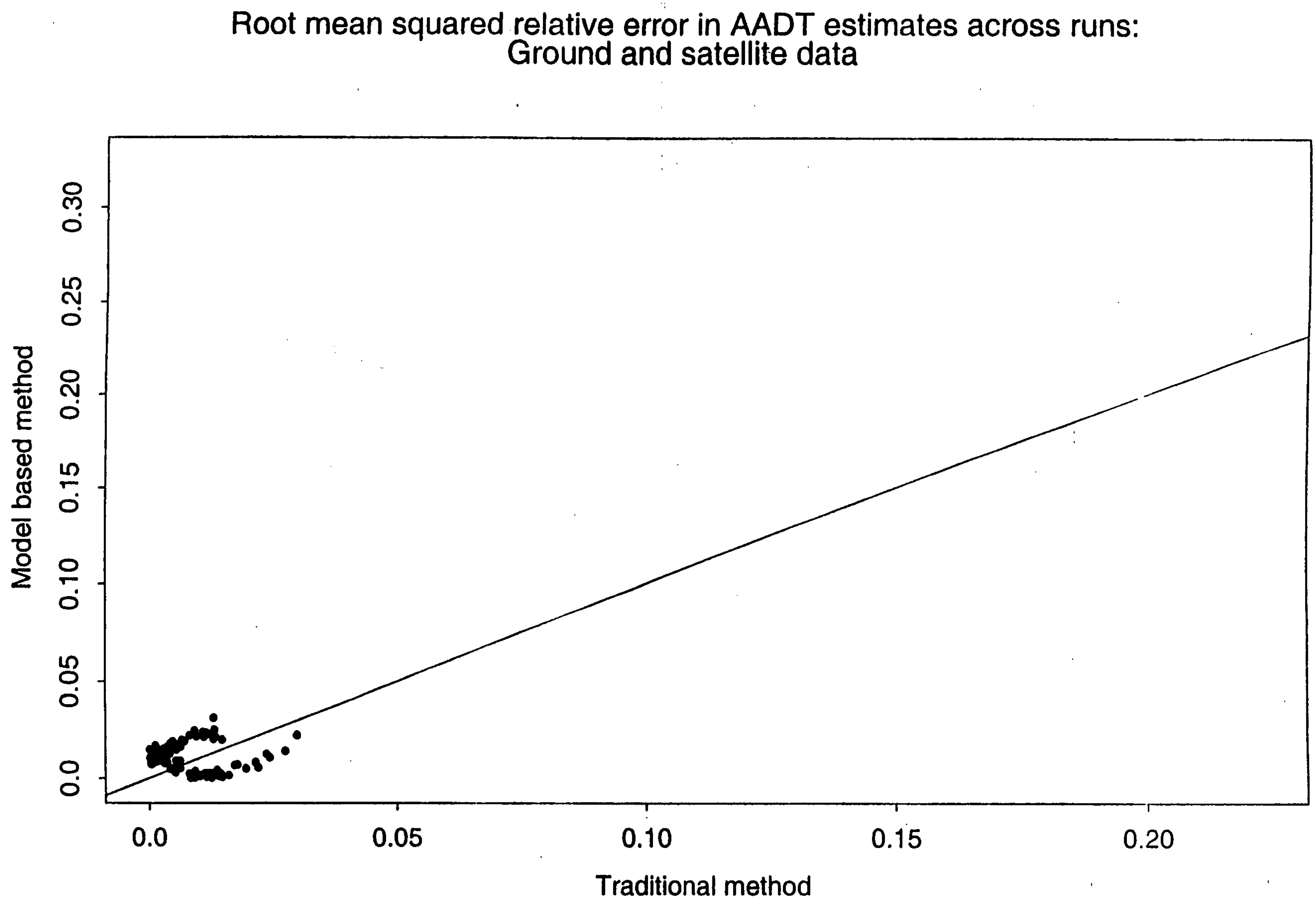


Figure F2. 25 moveable ATRs, satellite variance = 0.04, *ESC = 1.5*.

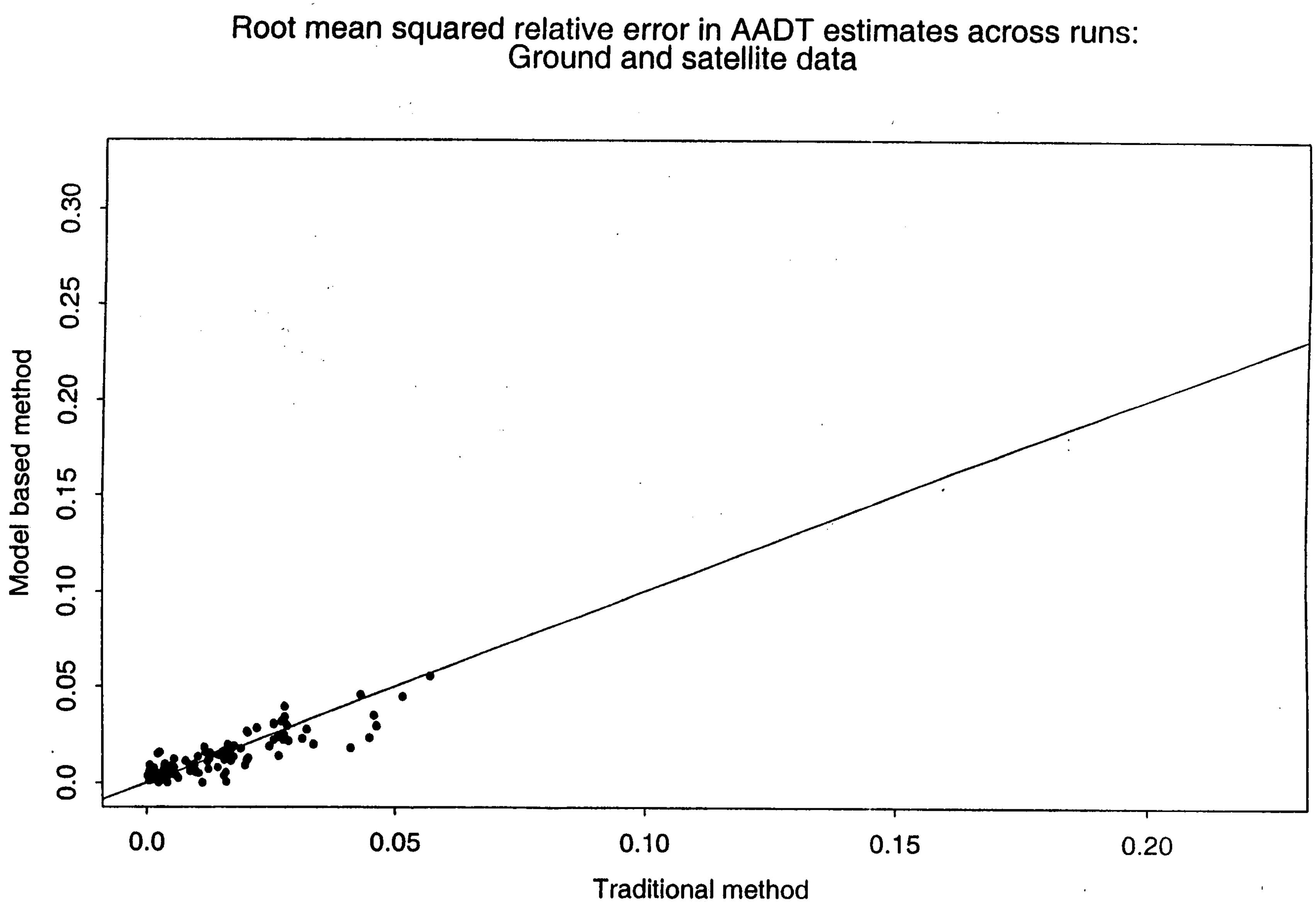


Figure F3. 25 moveable ATRs, satellite variance = 0.16, *ESC = 1.5*.

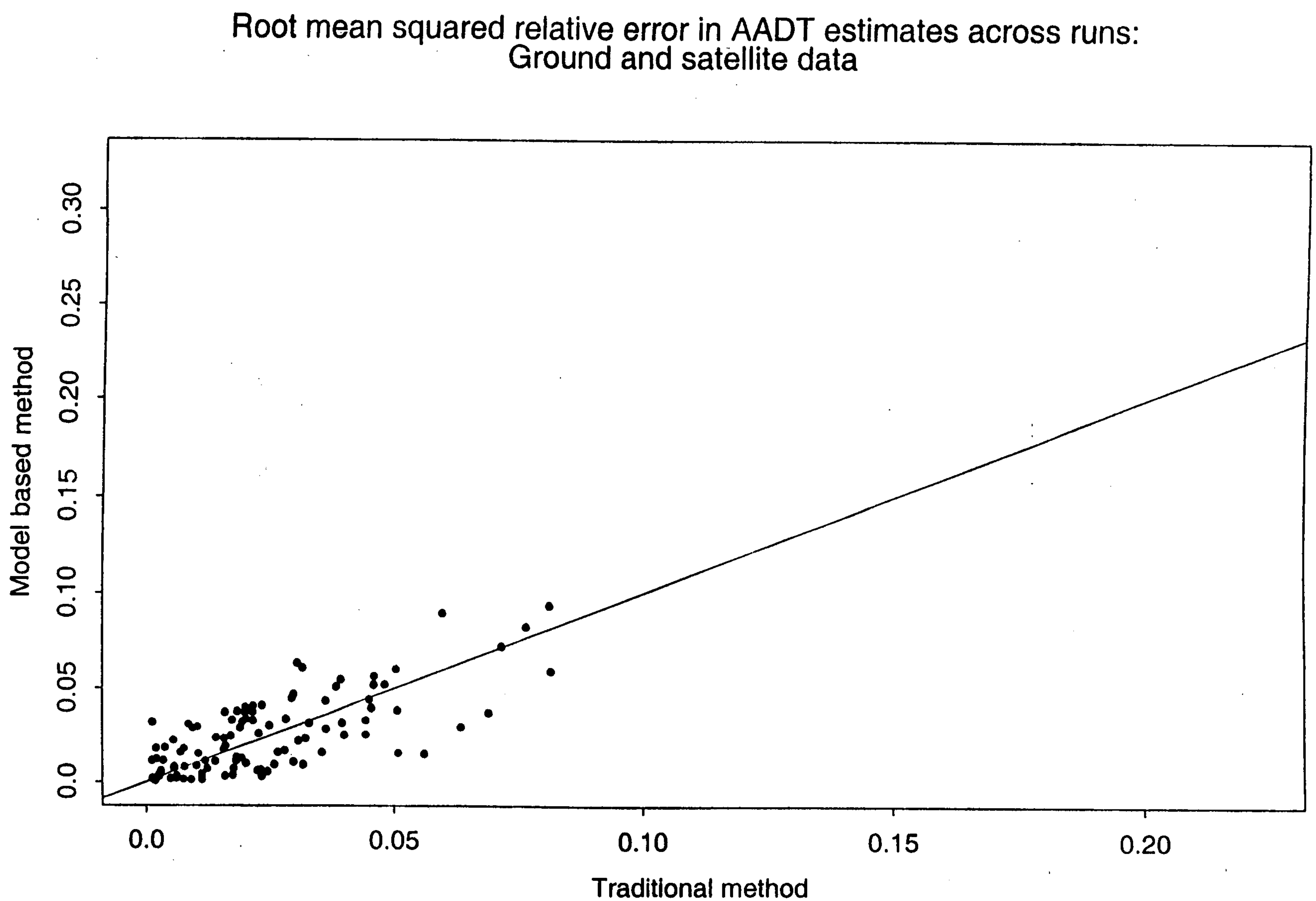


Figure F4. 25 moveable ATRs, satellite variance = 0.36, *ESC* = *1.5*.

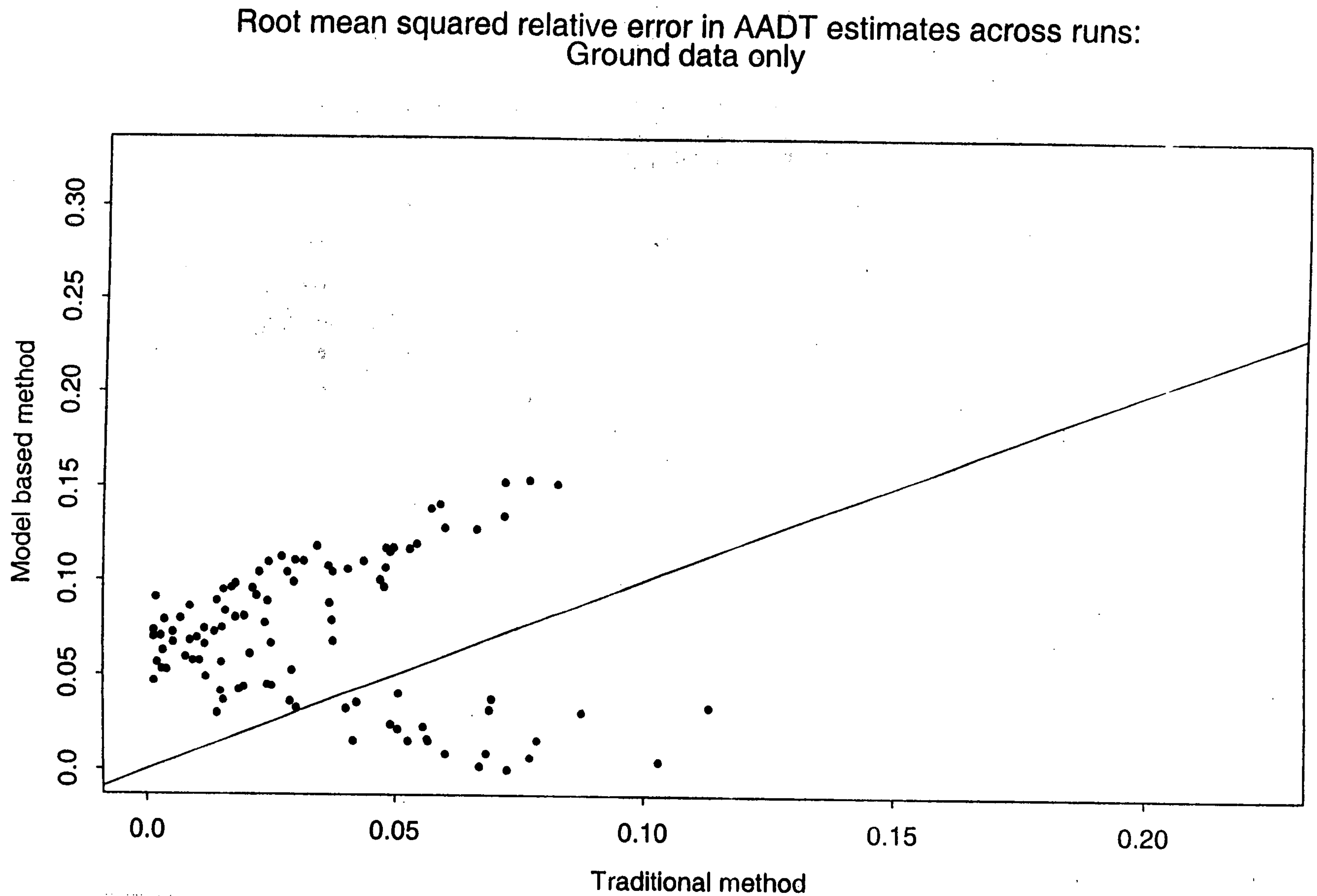


Figure F5. 50 moveable ATRs.

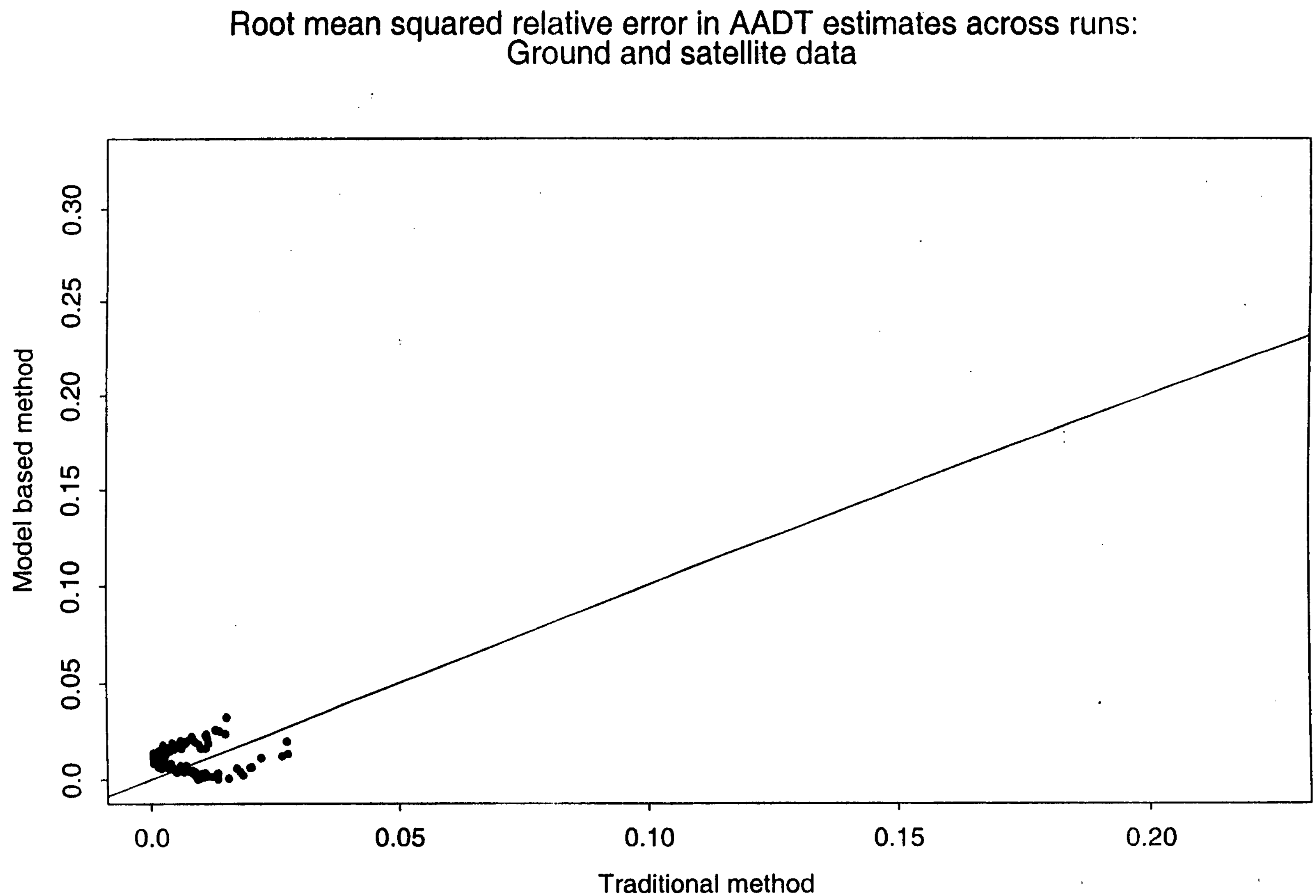


Figure F6. 50 moveable ATRs, satellite variance = 0.04, *ESC = 1.5*.

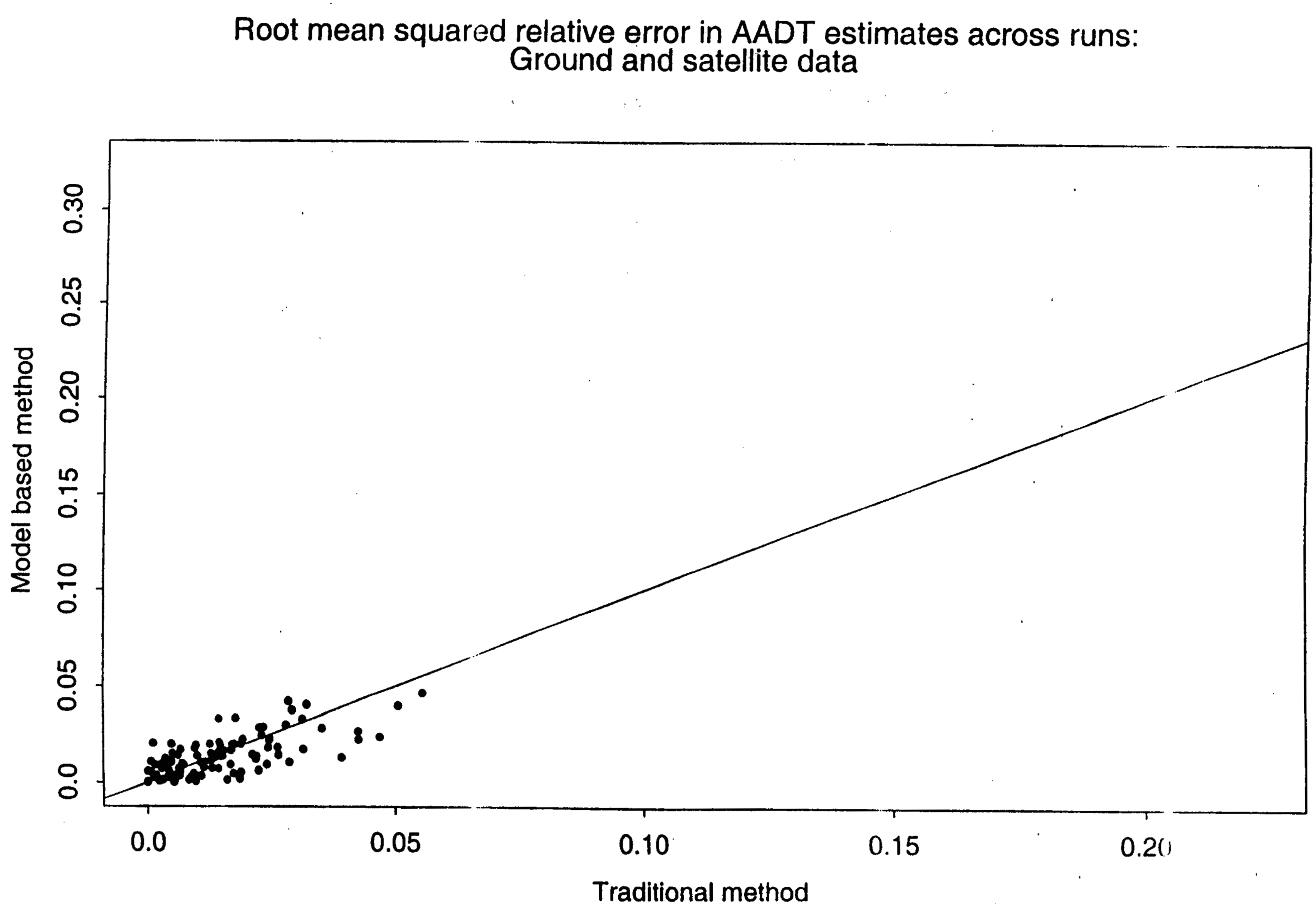


Figure F7. 50 moveable ATRs; satellite variance = 0.16, *ESC = 1.5*.

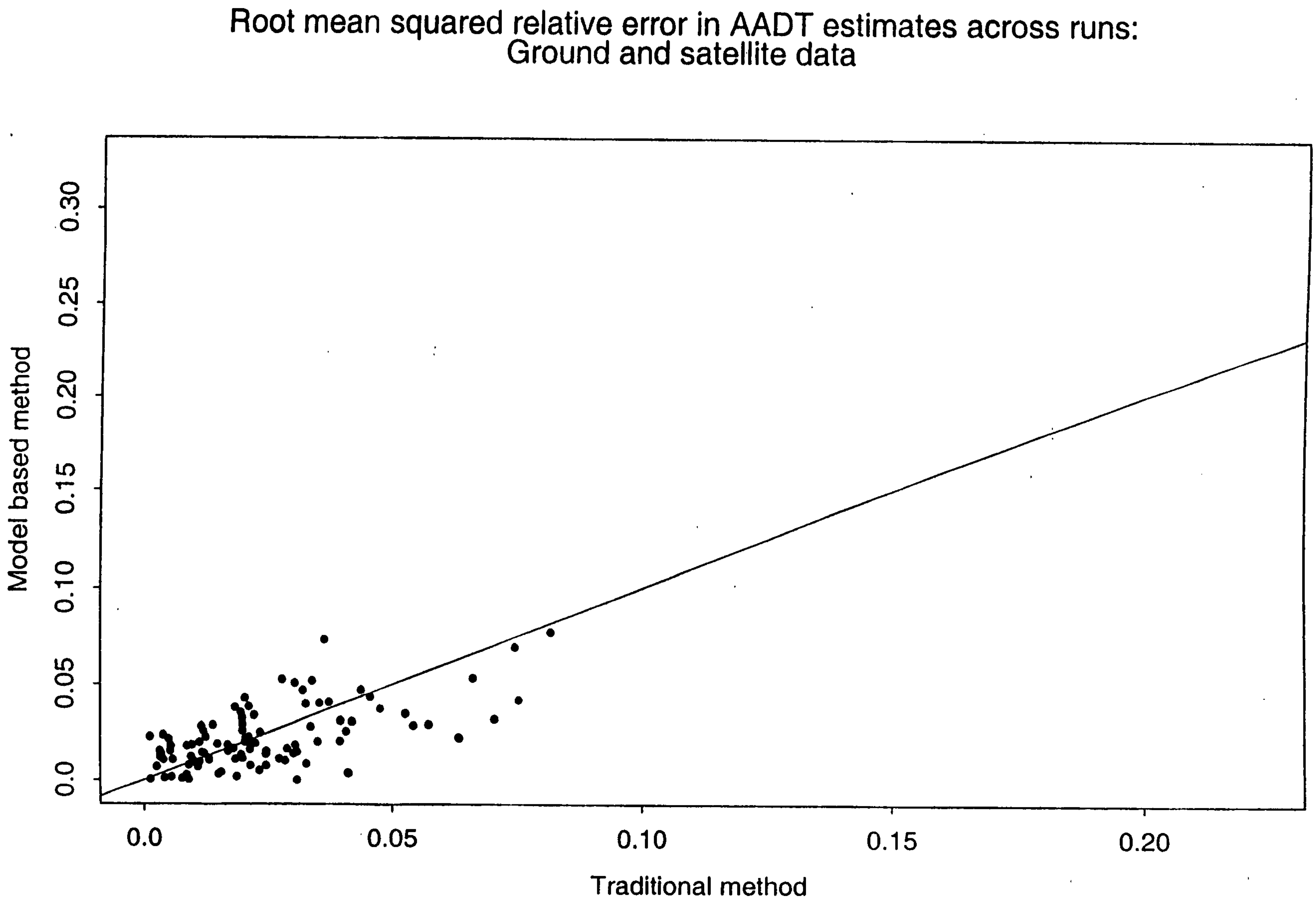


Figure F8. 50 moveable ATRs, satellite variance = 0.36, *ESC = 1.5*.