Final Report

TNW2009-16

Research Project Agreement No. 61- 7169

Self Calibrating Monocular Camera Measurement of Traffic Parameters

Daniel .J. Dailey

Department of Electrical Engineering
University of Washington
Seattle, Washington 98195-2500

A report prepared for

Transportation Northwest (TransNow)
University of Washington
135 More Hall, Box 352700
Seattle, Washington 98195-2700

in cooperation with

U.S. Department of Transportation
Federal Highway Administration

December 2009

TECHNICAL REPORT STANDARD TITLE PAGE

1. REPORT NO.	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO.
TNW2009-16		

4. TITLE AND SUBTITLE	5.REPORT DATE
Self Calibrating Monocular Camera Measurement of Traffic Parameters	December 2009
	6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S)	8. PERFORMING ORGANIZATION REPORT NO.
Daniel Dailey	TNW2009-16
9. PERFORMING ORGANIZATION NAME AND ADDRESS Transportation Northwest Regional Center X (TransNow) Box 352700, 129 More Hall University of Washington Seattle, WA 98195-2700	10. WORK UNIT NO.
	11. CONTRACT OR GRANT NO. DTRT07-G-0010
12. SPONSORING AGENCY NAME AND ADDRESS United States Department of Transportation Office of the Secretary of Transportation 1200 New Jersey Ave, SE Washington, D.C. 20590	13. TYPE OF REPORT AND PERIOD COVERED Final Research Report
	14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

ABSTRACT

This proposed project will extend the work of previous projects that have developed algorithms and software to measure traffic speed under adverse conditions using un-calibrated cameras. The present implementation uses the WSDOT CCTV cameras mounted along freeway segments and has an interface for automated camera calibration and traffic speed and speed variance measuring and recording. The calibration algorithm is implemented as an operator in a pipelined architecture using the Java Advanced Imaging package. The algorithm uses features found on the freeway, such as fog lines and lane markers, to calibrate the camera. Arterials have different features, such as turn arrows and stop bars, that can be used for calibration. This effort will develop algorithms that calibrates cameras based on common features found on arterials and will implement the algorithm as a Java operator so that it extends the capabilities of the software to arterials. The result will be a portable system that can function on both freeways and arterials with only limited infrastructure investment. The utility of this project is to leverage existing software to make traffic parameter measurements on arterials where there are no loops, and to calibrate traffic measurement and management devices using these measurements.

17. KEY WORDS	18. DISTRIBUTION STATEMENT
video image, detection, traffic parameters, camera	

19. SECURITY CLASSIF. (of this report)	20. SECURITY CLASSIF. (of this page)	21. NO. OF PAGES	22. PRICE
None	None	28	

Disclaimer

The contents of this report reflect the views of the authors, who are responsible for the facts and accuracy of the data presented herein. The contents do not necessarily reflect the views or policies of the Washington State Transportation Commission, Department of Transportation, or Federal Highway Administration. This report does not constitute a standard, specification, or regulation.

TABLE OF CONTENTS

Disclaimer **3**

Introduction **5**

Background **6**

A. Software **11**

B. Implementation **14**

Conclusions, Recommendations and Future Work **22**

References **23**

CHAPTER 1

Introduction

This project extends the work of previous projects that have developed algorithms and software to measure traffic speed under adverse conditions using un-calibrated cameras. [1,2] The past implementation uses the WSDOT CCTV cameras mounted along freeway segments and has an interface for automated camera calibration and traffic speed and speed variance measuring and recording. The past effort relied on the availability of the ITS Backbone to both provide a means to select cameras of interest and to transfer the digitized images from WSDOT Northwest region Traffic Management System Center (TSMC). The Backbone infrastructure is no longer in place due to financial considerations. A significant portion of the effort in this project is to make the previously developed software functional in the present environment for capturing video data.

The calibration algorithm is implemented as an operator in a pipelined architecture using the Java Advanced Imaging package. The algorithm uses features found on the freeway, such as fog lines and lane markers, to calibrate the camera. Arterials have different features, such as turn arrows, lane markers, crosswalks and stop bars, that can be used for calibration. This effort developed algorithms that calibrates cameras based on common features found on arterials and implemented the algorithm as a Java operator so that it extends the capabilities of the software to arterials. The result is a portable system that can function on both freeways and arterials with only limited infrastructure investment. The utility of this project is to leverage existing software to make traffic parameter measurements on arterials where there are no loops, and to calibrate traffic measurement and management devices using these measurements.

CHAPTER 2

Background

The presently installed set of cameras represents a large financial investment to all DOTs nationally. This research allows quantitative speed to be estimated from the existing camera pool and eliminates the need for additional, specially calibrated cameras and associated infrastructure. Furthermore, as cameras are deployed on arterial routes, this technology will allow quantitative traffic speed measurements to be estimated from those cameras; such measurements would otherwise be available only by installing additional, expensive traffic sensors (e.g., loops or radar).

Past projects have designed algorithms and constructed software that can use un-calibrated cameras to measure speed on freeways. Past projects have also built the Backbone network infrastructure that allows WSDOT cameras, both freeway and arterial, to be available for use as quantitative sensors. Innovative image processing techniques created, published and used in the prototype application include (1) the use of perspective straightening for image linearization [5], (2) an autocorrelation technique for lane stripe detection and estimation of the linear pixel/foot scale factor [7], and (3) a cross correlation technique used to estimate mean traffic speeds and direction of flow[3,6]. The approach is implemented for demonstration in prototype software. An example of the user interface is shown in Figure 1. The steps necessary to estimate traffic speed from video images are shown in Figure 2 and consist of (1) selecting a camera, (2) acquiring images, (3) calibrating the selected camera in the present position, if possible, and, if the camera calibration is successful, (4) making speed estimates of oncoming and receding traffic. Each of these steps is realized in a pipeline-like operation; in fact, the application is implemented by extending the Java Advanced Imaging package from SUN to create a pipeline member for each of the steps.

The previous projects were undertaken more than three years ago and the Backbone network infrastructure has changed. In addition, the software components build in the past efforts needs to be updated to compile and operate in the present operating and Java systems environment.

The present project devotes a significant portion of the approximately two months of available effort to modifying the software to: (1) be functional (compile and operate) in the current Java and Windows environments, (2) obtain images from the arterial cameras using the Axis encoding device in the STAR lab, (3) identify useful arterial locations to develop algorithms for calibration, (4) developing additional software functionality to record camera

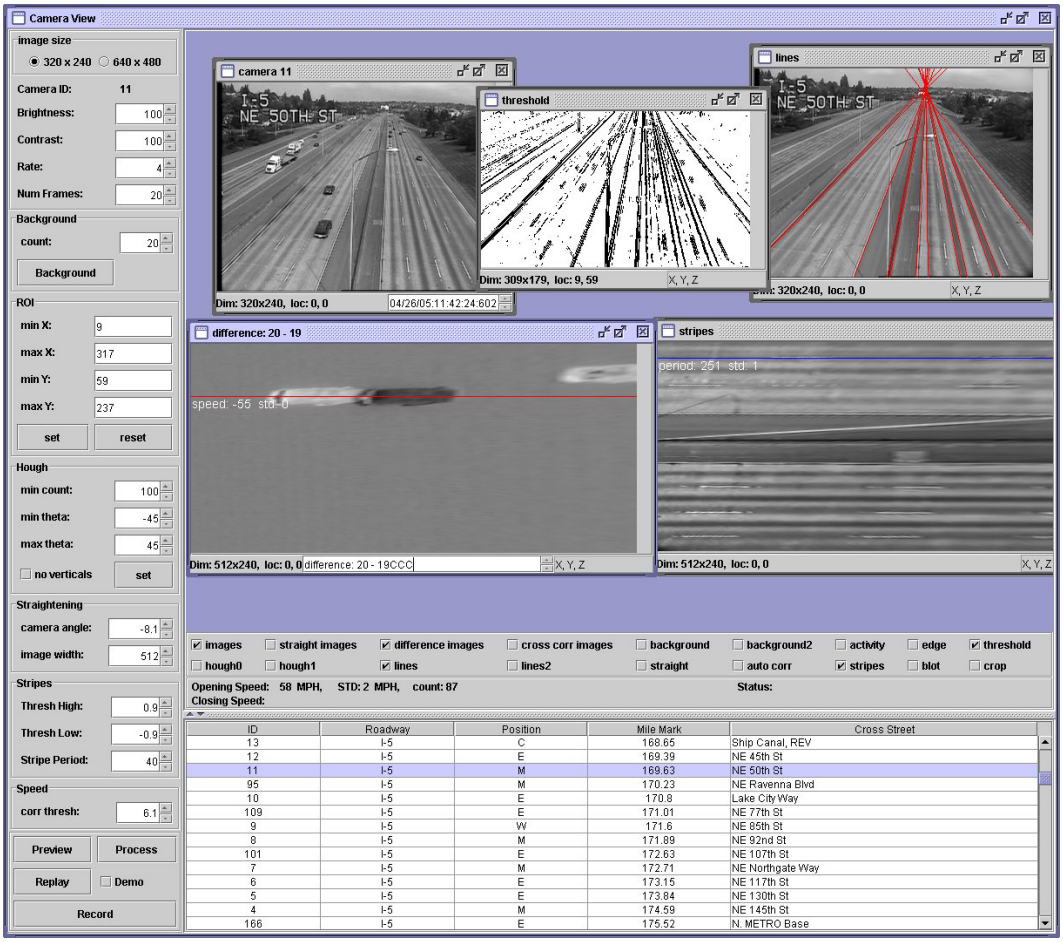


Figure 1 Camera calibration and speed measurement application.

data from the intersections using the Axis encoder

To automatically calibrate the camera, the image needs to be “straightened” or warped, so that pixels have the same effective size throughout, and scaled, so that the size of each pixel can be expressed in feet. The straightening operation involves finding a vanishing point in the image, and the scaling operation requires finding lane striping. The three phases for implementing calibration are (1) highway line detection, see Figure 3, (2) computation of the

vanishing point and image straightening transformation, see Figure 4, and (3) computation of the image-to-highway scale factor (feet/pixel). The algorithm is implemented as a series of operations that are shown in Figure 5. Once the camera has been calibrated, a cross correlation between sequential images is used to estimate traffic speed. This cross-correlation method is robust in the face of high density traffic with much occlusion of vehicles, a situation in which other camera-based algorithms fail. This method has analytical bounds on the accuracy of the speed measurement as shown in [1,7] that guarantee the performance of the speed estimation algorithm.

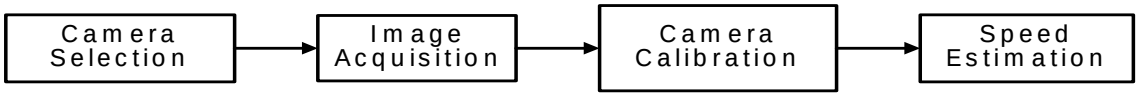


Figure 2 Steps to perform speed estimation with an un-calibrated camera

Figure 3 Line and vanishing point identification

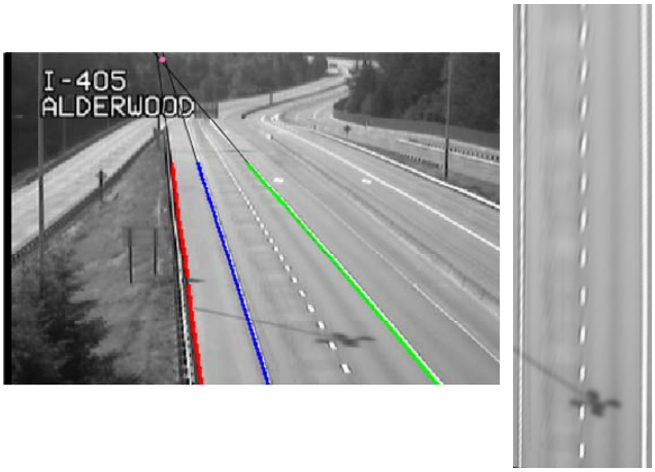


Figure 4 Image Straightening

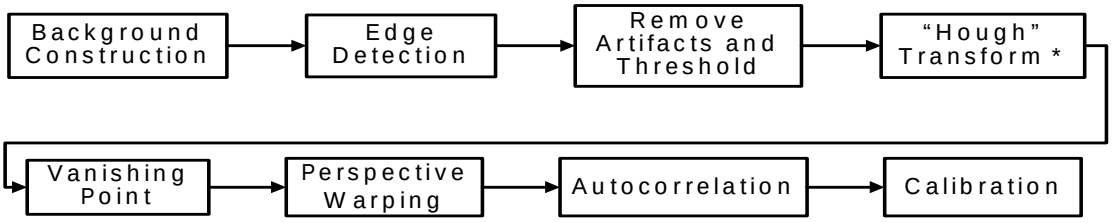


Figure 5 Steps used in calibration process.

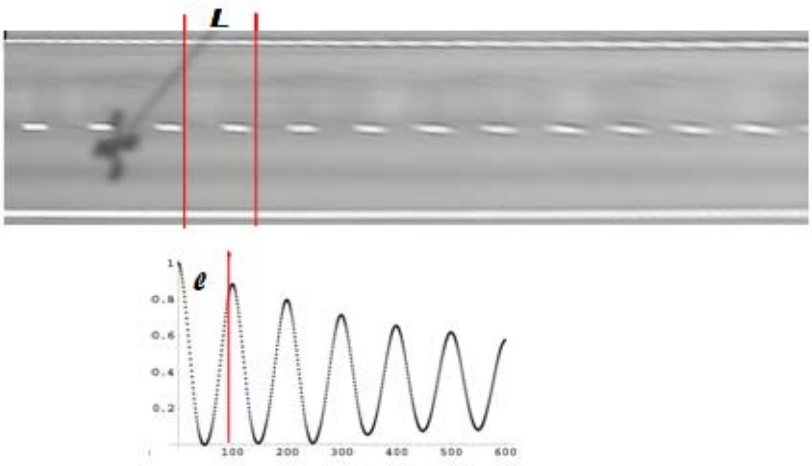


Figure 6 Obtaining scaling factor from roadway.

This project expands the calibration algorithm described in [1] to be used on arterials. Arterial data needs differ from those on the freeway. If the camera is mounted mid-block, it may be used as a speed senor just as on the freeway. However, many cameras are mounted at or near intersections and need to measure several parameters, such as queue length, vehicle classification, vehicle lengths, turning movements, occupancy and speed, while being able to tilt pan and zoom. These requirements lead to a need to expand the number of roadway features for calibration.

The arterial fog and centerlines are widely available in imagery from regional arterials. These can be used to identify a vanishing point, which in turn allows for warping/straightening of the image, as in Figure 4. This straightened image allows for features, such as vehicles, to be identified in a coordinate system that has a linear relationship with the ground plane, e.g. rectangular cars are rectangular in the image and of equal size throughout the image. As a result, measures that primarily use ratios such as occupancy, number of vehicles in a queue, turning movements, and vehicle classification can be done accurately using the straightened images. Algorithms to perform these measurements and implementation of those algorithms in the pipeline are shown in Figure 2. . These measurements can be enhanced to provide accurate world length measurements if the image can be completely calibrated.

In Figure 6, the lane stripe period found in the straightened image provides a scale factor for the calibration on the freeway. This was chosen because it is a feature nearly always found on regional freeways. This project will expand the automated calibration to include features such as turn arrows, crosswalk lines, and transit symbols for scaling. These common features will allow complete scaling of the straightened image. Once the image is completely scaled and straightened features in the image can be measured with a known, and mathematically provable, accuracy. Queue lengths measure in feet, absolute speed and vehicle lengths in feet can accurately be measured.

CHAPTER 3

PROJECT ACTIVITIES

A. Software

The past projects developed a distributed Java application for traffic video image acquisition, camera calibration and speed estimation. There was a server program "*Capture*", running on a computer placed on the WSDOT Traffic Management System Center's (TSMC)local network, with access to the CCTV camera video switch. This program both controls the switch, as well as grabbing and serving video image sequences over the ITS Backbone. A client application program "*AutoCalSpd*", on the UW-ITS local network, requests and processes the image sequences. Communication between server and client is through a controlled proxy program which prevents unauthorized users from accessing the server. The *AutoCalSpd*' had information on the relationship between physical camera locations and the port on the TMSC and automated the camera selection over the ITS Backbone. Figure 7 shows the previous data flow to access traffic video.

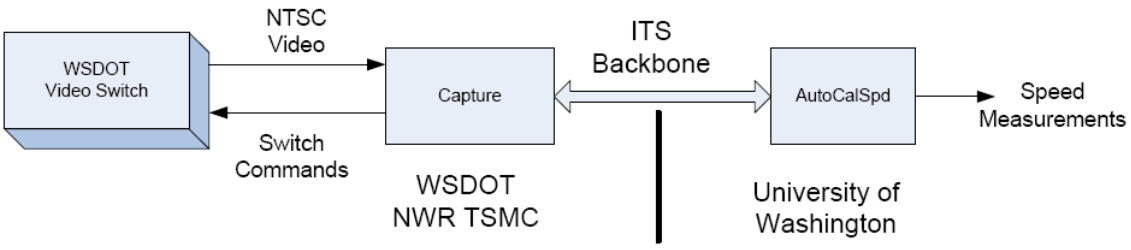


Figure 7 Previous network implementation.

Due to the replacement of the WSDOT Traffic Management System at WSDOT's Northwest Region the structure and methodology of accessing the cameras was changed. The ITS Backbone was turned off and a dedicated fiber from WSDOT to StarLab at the UW was put into place as shown in Figure 8. In the new configuration the camera selection is done asynchronously outside the AutoCalSpd application and the images are captured and digitized by the Axis encoder. This change in structure requires the code that acquires images to be completely rewritten to be able to obtain, accept and process images from the Axis encoder. This rewriting of the code consumed a significant portion of the project, but.

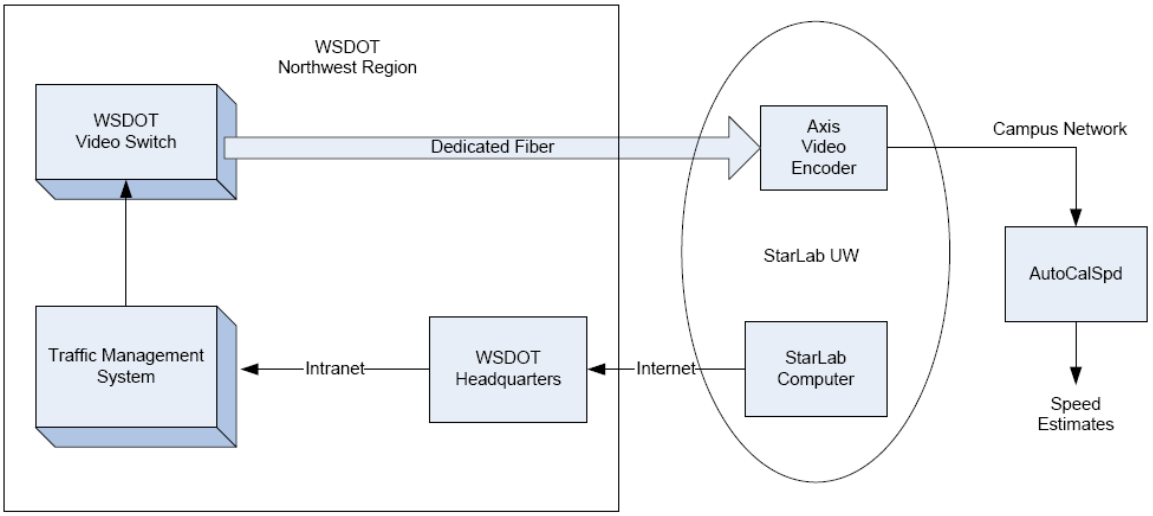


Figure 8 Present network setup

has been accomplished. The configuration file for the application can now have the command “source : axis” so that the image processing application can use images from the Axis encoder over a tcp-ip network. An example of the configuration file can be seen in Figure 9. This software change will allow the AutoCalSpd application to accept data from any NTSC source that is plugged into an Axis encoder and provide a standardized published interface to access data from any agency that uses Axis encoders.

```
#SERVERHOST : VIDEO

SERVERHOST : LORRY

#IF USING AN SSH TUNNEL

#SERVERHOST  :128.95.204.131

#SERVERPORT : 80

SERVERPORT : 9006

#CAMLIST : CAMLIST

#CAMLIST : CAMERA.REP

CAMERAS : CAMERAS_9-13-2006.TSV

CONTROLS : CONTROL_PARMS.CSV

MAXNUMBIGFRAMES : 20

# FRAMES SAVED FROM PC VIDEO CARD

#SAVEFRAMES : TRUE

#SAVEFRAMESFILE : FRAMES_23JUL2009

# IMAGES SAVED FROM AXIS BOX

#SAVEIMAGES : TRUE

#SAVEIMAGESFILE : IMAGES_24JUL2009

SOURCE = DEMO

#DEMOFRAMESFILE : FRAMES_9MAR2006

#DEMOFRAMESFILE : FRAMES_23JUL2009

#SOURCE = AXISDEMO

#DEMOIMAGESFILE : IMAGES_23JUL2009

#SOURCE = AXIS
```

Figure 9 Configuration file.

B. Implementation

This section presents the implementation of the image processing client application AutoCalSpd. There are two main parts to the application: the image processor that performs the calculations for calibration and speed estimation, The image processor makes use of the Java Advanced Imaging (JAI) framework. The graphical user interface is implemented using the Java Swing GUI toolkit. The GUI allows camera selection, parameter tuning and visibility into the various image processing stages. There is an interactive mode of operation in which the user supervises the calibration, and there is an automatic mode in which speed data is continuously recorded into a file.

Graphical User Interface

The user interface is implemented using the Java Swing GUI toolkit. It consists of three principle components: a control panel, an image display desktop panel and a camera selector panel. A screenshot of the GUI is shown in Figure 1. However, in the case of using the application with an Axis encoder, the present default network, the camera selection controls do not function and the camera selection need be done asynchronously using software from WSDOT.

When the application is launched, the default camera is the one plugged into the Axis encoder. A (320 X 240 pixel) snapshot image of the field of view is shown in the display panel. All calibration parameters are set at their default values and the ``try calibrate" toggle is set, but no calibration will be attempted until the ``load images" button is pressed. The ``try speed" toggle is disabled since no image-to-highway scale factor is known yet. The user may switch cameras, increase the image size to (640 X 480), and preview images at any time. For calibration purposes, a camera should be selected with the following properties: The camera should view approaching and/or receding traffic, that is, it should point downward and along the highway rather than across it. The view should show straight highway lines in the bottom half of the image as well as lane stripes. The view should be mostly unobstructed by overpasses, large overhead signs, divergent lanes of traffic, etc. (This condition may be relaxed in some cases by editing certain control parameters during the calibration process,

for example moving the baseline of the ROI further up into the image, narrowing the angular limits of the Hough transform, or increasing the height of the straightened image.) Under these conditions a successful calibration is likely. To effect a calibration, the user selects the image size, number of images to acquire and the frame rate, and then presses the ``load images" button. This results in a request being sent to the Axis encoder. The response from the server will be a sequence of time-tagged images, or frames, the first of which is presented in a ``captured frames" viewer in the image display desktop. The user can cycle through the images using ``spinner" buttons attached to the bottom of viewer. Since the ``try calibrate" toggle button is selected, a calibration process is attempted. The user may view images produced at various stages of the process by pressing labeled buttons under the desktop. This is useful for confirming the validity of a calibration or diagnosing and maybe correcting a failure.

If the calibration fails, an error message pops up indicating where the failure occurred: either no vanishing point was found, or no stripes were detected. The desktop may be used as a diagnostic tool in these cases. Visibility into the image processing stages prior to the failure point may suggest parameter changes that could lead to a successful calibration. For example, if the vanishing point could not be found it is useful to view the ``lines" image which shows detected lines superimposed on the background image. If too many or too few lines are shown, the user may edit the Hough parameters and force a reactivation of the calibration process starting at the Hough stage. If stripes could not be found it is useful to view the ``straightened background" image. This may show that not enough stripes are present, in which case the length of the straightened image may be increased and the calibration process reactivated beginning with the straightening step. If the stripes are faint, the stripe detection thresholds may be lowered and the calibration process reactivated beginning with the stripe detection step.

If the calibration process is successful, a status message under the desktop indicates “successful calibration" and the “try speed" toggle button is enabled. However, before activating a speed computation, the user should view the “stripes" image to double check that road stripes were actually found rather than some other periodic structure such as

construction barrels. (Also, if an insufficient number of image frames are collected, traffic can appear as a periodic structure in the background.)

Selecting ``try speed" will start a speed estimation process using default values for the correlation threshold and the stripe period (40 feet for freeways). When this process completes, a status message under the desktop indicates “speeds computed" and the current approaching and receding traffic speed estimates are displayed also just below the desktop. If no speeds are shown, then either there is no traffic (which can verified by spinning through the “captured frames" viewer) or the cross correlation threshold is too high. The user can edit this parameter and reactivate the speed computation. If the speeds appear unreasonable, then the default stripe period may be wrong (some arterial stripes are spaced at 12 and 15 foot intervals).

Once satisfied with the camera calibration and speed parameter settings, the user may continue interactively to load images, and as long as “try speed" is selected, speeds will be computed. The automatic mode of operation may be enabled by pressing the “record" button. The program will repeatedly load images, compute speed reports, and append them to a file. A test is performed on each cycle to determine if the camera calibration (scale factor and straightening transformation) are still valid. If not, a popup alerts the user that action needs to be taken: either select “try calibrate" or select a new camera and start over.

Image Processor

The image processor is the heart of the AutoCalSpd application. It consists of an image processing graph (see Figure 10) whose nodes implement the various algorithms described in [1] and methods for manipulating the graph. In particular, the processor manages the loading of source images and the rendering of the graph according to the controls and parameter settings imposed by the user. The processor can be in one of two states: calibrate or compute speed, and regulates rendering of the graph accordingly.

The image processor is implemented in Java and relies heavily on the Java Advanced Imaging (JAI) application programming interface. The JAI provides a set of basic image processing operators and a framework for defining and registering custom operators. Every

operator stores an operation name, a *ParameterBlock* containing sources and parameters, and a *RenderingHints* which contains image rendering hints, such as image size and format.

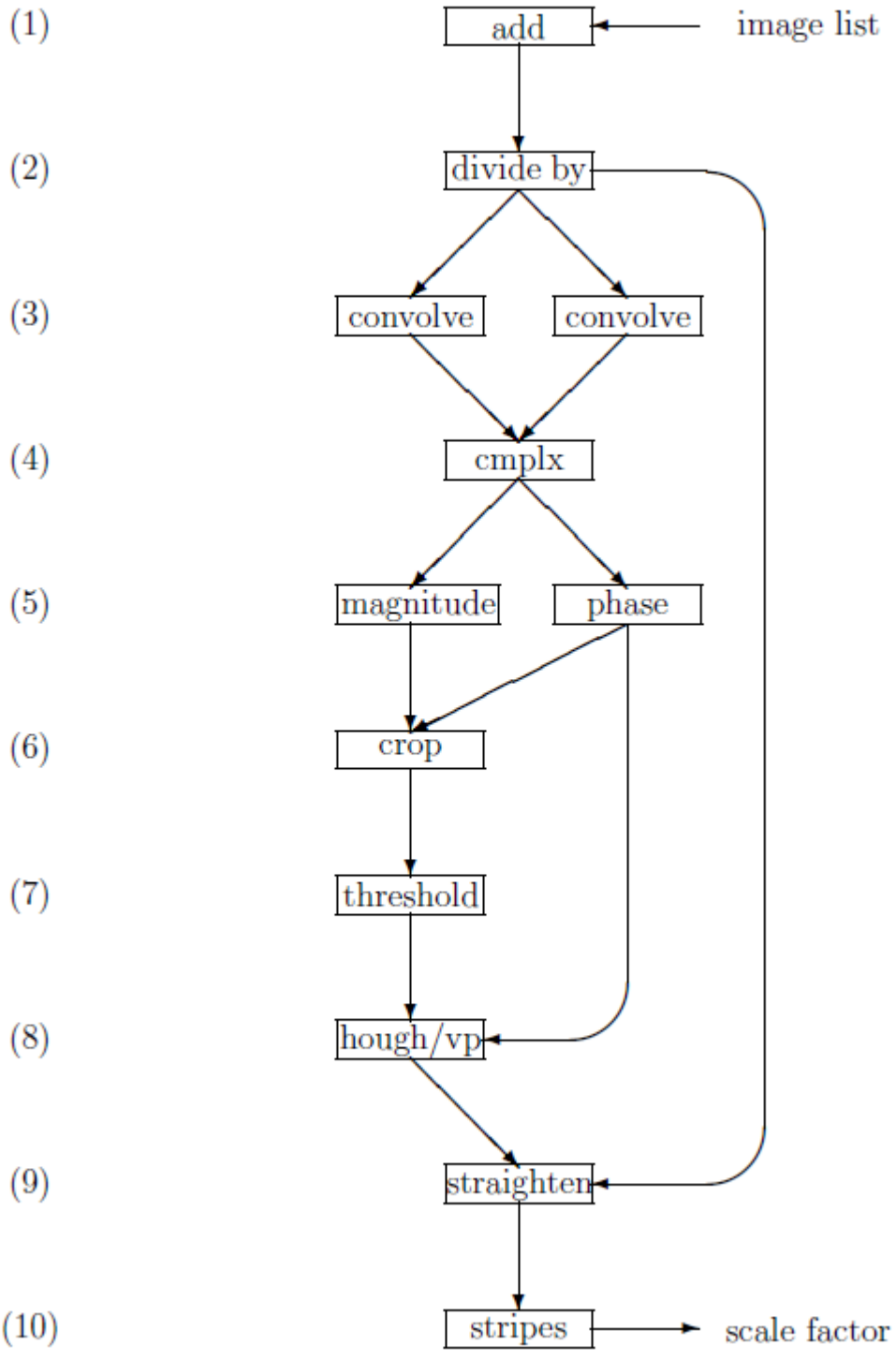


Figure 10 Image processing graph.

Programming in JAI generally involves constructing an image processing chain, or more generally, a directed acyclic graph (DAG) whose nodes are operators. This is useful in that a chain or DAG may be manipulated dynamically and rendered multiple times. Thus for example the same chain of operations may be applied to different images or the parameters of certain operations in a chain may be modified interactively. It is important to note that image rendering adheres to the pull model, that is, a node is rendered only when there is a request for actual pixel data.

The DAG

Figure 10 shows the basic DAG for the image processor. Operators at levels (1) - (5) are provided with the JAI distribution, while operators at levels (6) - (10) are custom. The graph is constructed during the initialization phase of the program. The operators are created and linked together in top down, left to right order using default parameter settings. Once the DAG has been constructed, the user can request the processor to load source image data or make changes to various operator parameters.

Operators (1) and (2) taken together implement the ``background" algorithm discussed in [1]. The image layout parameter for the “add" operation specifies that the input data type (byte) should be cast to type (double) so that the correct sum is computed. The parameter for the “divide by" operator must be set to the length of the input image list.

Operators (3) - (5) implement the algorithm for computing the gradient in polar form. Parameters for the two “convolve" operators are the appropriate Sobel masks for computing the gradient components in Cartesian form. The “cmplx" operator simply combines the two components into a single complex image, while the “magnitude" and “phase" operators compute the polar components of the gradient. The “crop" operator (6) implements the “crop magnitude" algorithm described in [1]. Parameters are the bounds of the rectangular ROI and the gradient angle threshold.

The “threshold" operator (7) implements the “edge detect (threshold)" algorithm described in [1]. The threshold level is computed automatically using Otsu's method. The parameters for this operator consist of two switches: one to enable double thresholding and one to enable

non-maximal suppression (see [8]). Both of these switches are currently off. The "hough/vp" operator (8) implements the algorithms for computing Hough maps and the vanishing point detection algorithm from [1]. Parameters consist of the Hough threshold, two domain angle limits and a "no verticals" switch. This operator is unique in that its imagery is not used downstream. It computes the vanishing point and baseline needed by the "straighten" operator which is its sink.

The "straighten" operator (9) constructs the straightening warp discussed in [1]. This depends on the vanishing point and baseline computed above. The warp is applied to the background image computed at level (2). Parameters are the depression angle (ϕ) and the height of the straightened image (512 or 1024). Since image data is stored in row major order and autocorrelation is to be performed on columns, we rotate the image (90^o) for ease of data access.

The "stripes" operator (10) implements the autocorrelation algorithm for determining the scale factor discussed in [1]. Parameters are the upper and lower autocorrelation thresholds for stripe detection.

Implementation Summary

The application presented provides a Graphical user interface based on the Java Swing components. The selection of this set of components makes the GUI both portable and capable of being easily modified by a Java programmer.

The Image Processor implements the image processing in a pipeline model of components where half are taken from the Java Advanced Imaging framework and half are custom components. This architecture allows individual steps in the algorithms to be modified or replaced in a component wise structure. This allows for experimenting with a variety components based on the information expected in the video images.

Conclusions and Recommendations

In this project a java based image processing application developed in previous projects has been updated and modified for the new data sharing environment at the UW and WSDOT. The previous project required a video digitizer connected to the WSDOT video switch port (or at least to a fiber connected to a switch port) to capture video. The modifications made in this project allow the application to connect to any video stream available through the manufacturer Axis's generic conversion box. As a result the application can be used by more agencies and researchers and is not limited to the existing, and old, capture hardware and software.

The component structure of the application for both the GUI and Image Processor allow for a clear programmatic method to make substantial changes in both the look and feel as well as the underlying algorithms while keeping the overall general framework.

The present setup for accessing WSDOT video streams is substantially more difficult to use than in the past implementations. The ability to change the camera selected by the video switch to be displayed on the allocated video port now requires substantial manual intervention using custom software created at WSDOT whereas the previous backbone allowed for software in the GUI implementation to select cameras with a mouse click. A programmable application user interface (API) to the WSDOT custom software would vastly improve researcher's access to the video stream from the cameras without impacting network security for WSDOT. For example, as a result of the new video access framework the majority of this project was expended on interface activity rather than new algorithmic efforts.

References

[1] CCTV Technical report - Phase 3, D.J. Dailey and F.W. Cathey, *Washington State Transportation Center - TRAC/WSDOT, Final Technical Report WA-RD 633.1*, 41 pages, January 2006.

[2] Automated use of Un-Calibrated CCTV Cameras as Quantitative Speed Sensors - Phase 3, D.J. Dailey and F.W. Cathey, *Washington State Transportation Center - TRAC/WSDOT, Final Research Report WA-RD 635.1*, 27 pages, January 2006.

[3] A Cross-Correlation Tracking Technique for Extracting Speed from Cameras Under Adverse Conditions, T.N. Schoepflin and D.J. Dailey, *Transportation Research Record, 1867*, 36-45, 2004.

[4] Dynamic Camera Calibration of Roadside Traffic Management Cameras, T.N. Schoepflin and D.J. Dailey, *IEEE Transactions on Intelligent Transportation Systems*, Vol. 4: No. 2, pp. 90-98, June 2003.

[5] Mathematical Theory of Image Straightening with Applications to Camera Calibration, F.W. Cathey and D.J. Dailey, *Proceedings of the IEEE 9th International Conference on Intelligent Transportation Systems*, Toronto, Canada, 18-20 September, 2006.

[6] A Novel Technique to Dynamically Measure Vehicle Speed using Un-calibrated Roadway Cameras, F.W. Cathey and D.J. Dailey, *Proceedings of the IEEE Intelligent Vehicles Symposium*, Las Vegas, NV, 6-8 June, 2005.

[7] One-Parameter Camera Calibration for Traffic Management Cameras, F.W. Cathey and D.J. Dailey, *Proceedings of the IEEE 7th International Conference on Intelligent Transportation Systems*, Washington, DC, 4-6 October, 2004.

[8] Sonka M., Hlavac V. and Boyle R., Image Processing, Analysis, and Machine Vision PWS Publishing, Brooks/Cole Publishing Company, Pacific Grove, California, 1999

APPENDIX I – ARTERIAL AND STATE ROUTE DEMONSTRATION

This appendix provides some visual demonstrations of the application operating on arterial and other State Route (SR) roadways. The following roadways are used for demonstration:

US2 and Home Acres Rd

SR104 and 19th Ave NE

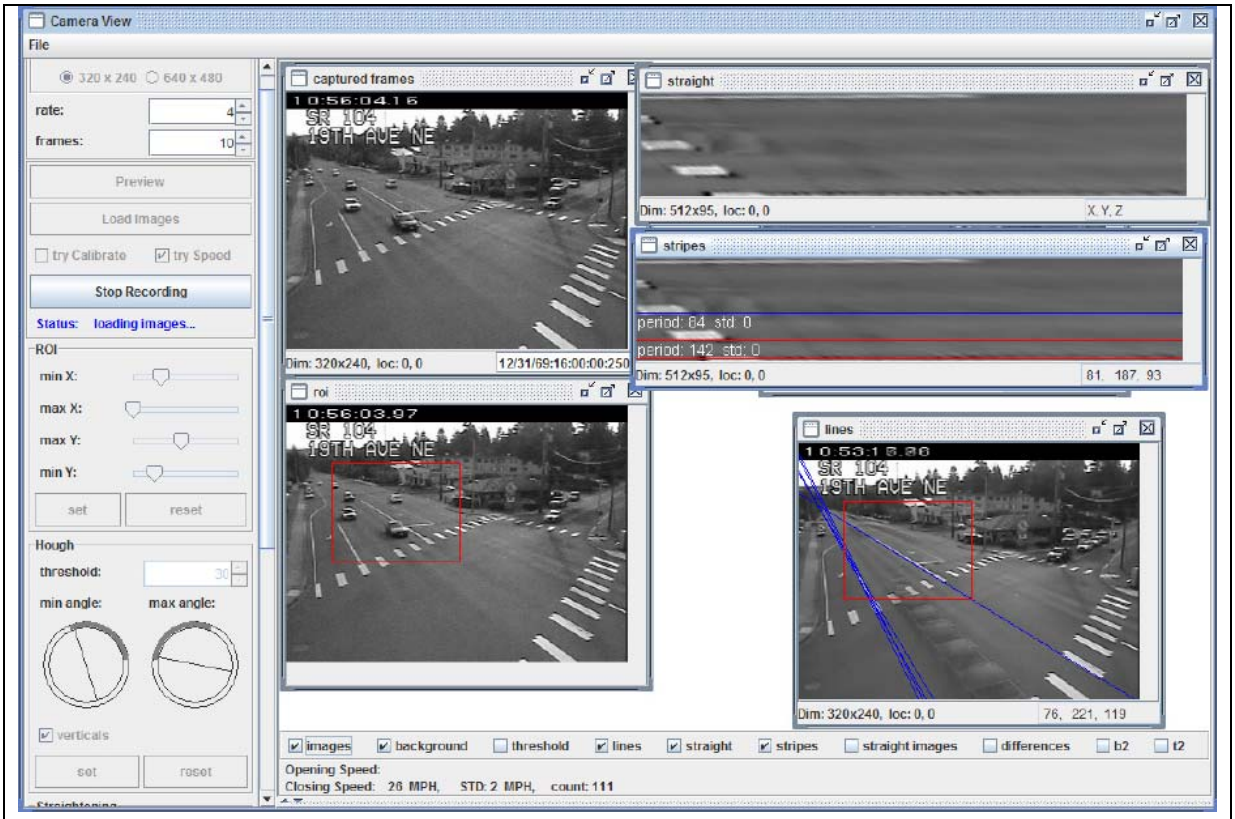
SR104 and Meridian Ave. N

SR161 and Military Road

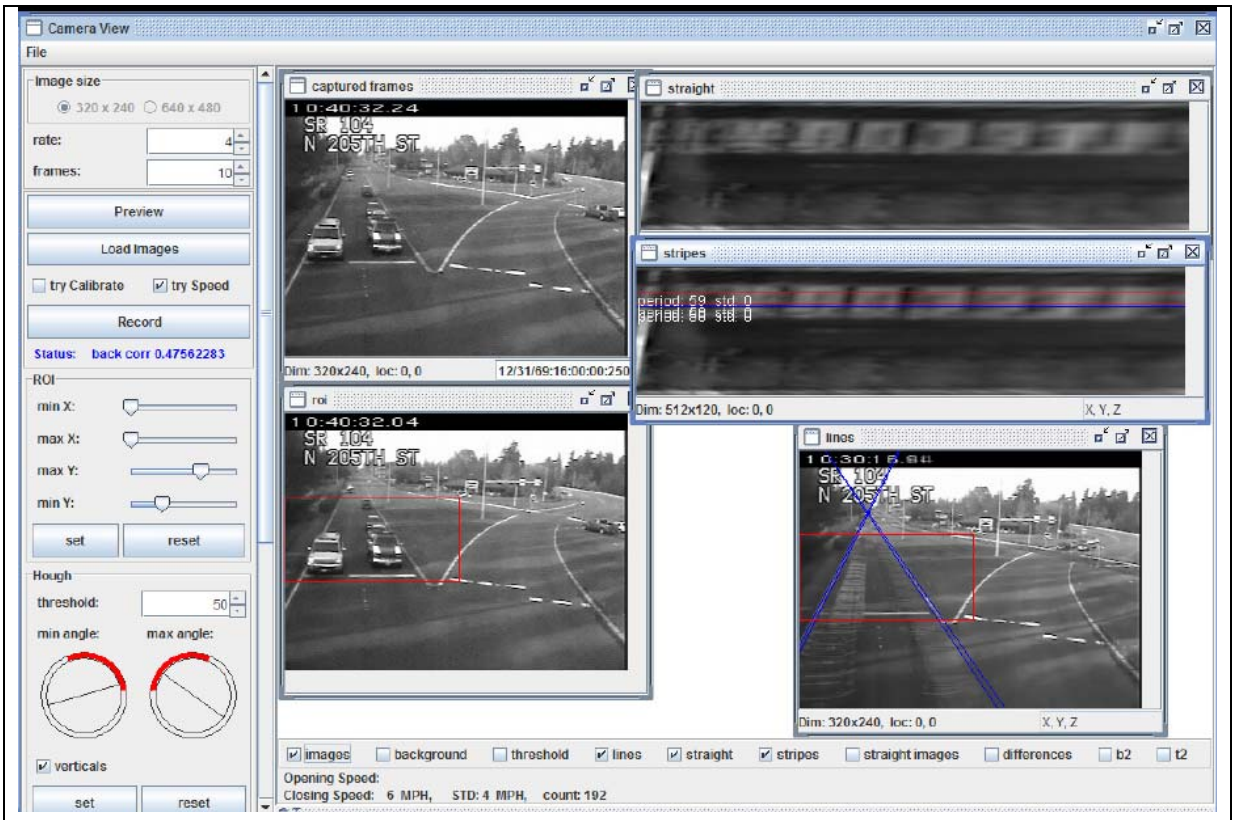
SR167 and 23rd St. SW

The speed estimate, along with the standard deviation appears at the bottom center of the imagery.

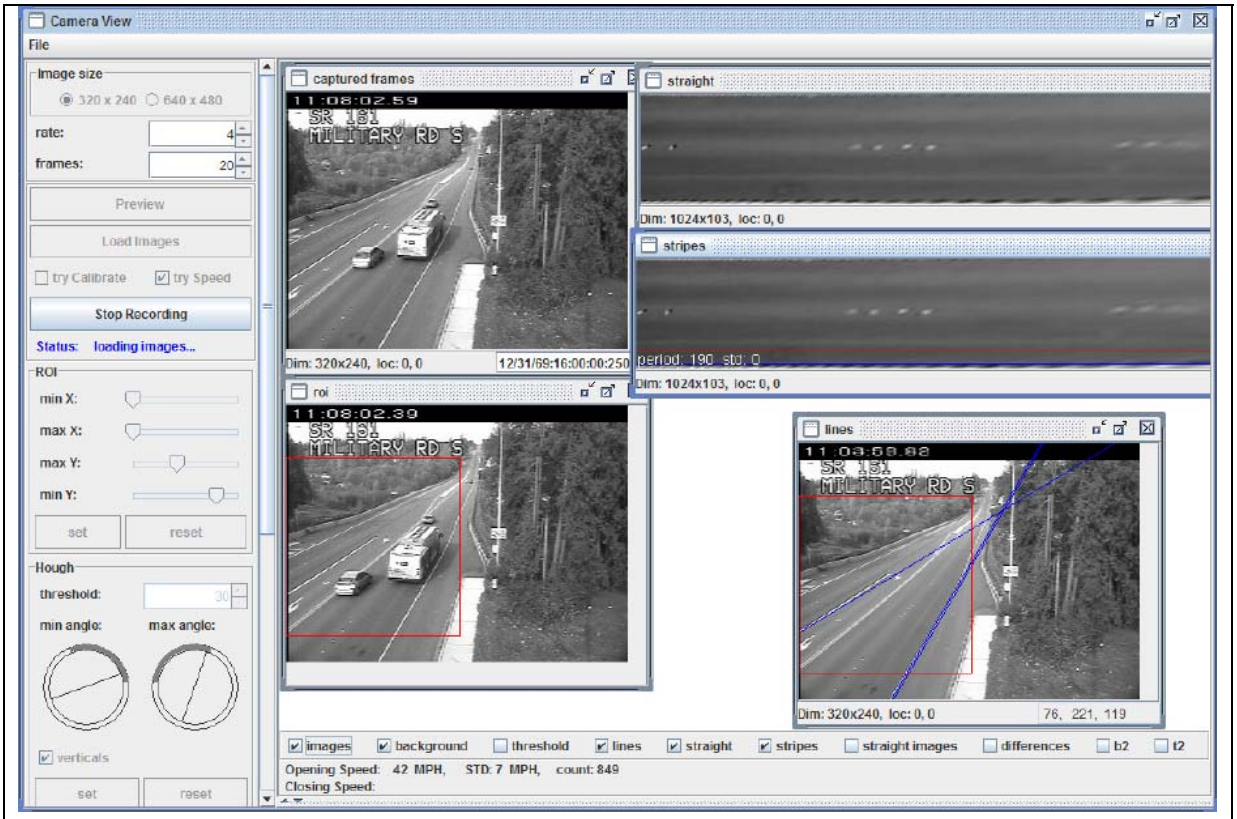
US2 and Home Acres Rd



SR104 and 19th Ave NE



SR104 and Meridian Ave. N



SR161 and Military Road

SR167 and 23rd St. SW