ON-LINE IMPLEMENTATION OF A FUZZY NEURAL RAMP METERING ALGORITHM

WA-RD 442.1

Research Report
August 1997

Washington State Department of Transportation

Washington State Transportation Commission
Planning and Programming Service Center
in cooperation with the U.S. Department of Transportation
Federal Highway Administration

TECHNICAL REPORT STANDARD TITLE PAGE

1. REPORT NO.	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO.
WA-RD 442.1		

4. TITLE AND SUBTITLE	5. REPORT DATE
On-Line Implementation of a Fuzzy Neural Ramp Metering Algorithm	August 1997
	6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S)	8. PERFORMING ORGANIZATION REPORT NO.
Cynthia E. Taylor and Deirdre R. Meldrum	
9. PERFORMING ORGANIZATION NAME AND ADDRESS	10. WORK UNIT NO.
Washington State Transportation Center (TRAC) University of Washington, Box 354802 University District Building; 1107 NE 45th Street, Suite 535 Seattle, Washington 98105-4631	11. CONTRACT OR GRANT NO. Agreement T9903, Task 70
12. SPONSORING AGENCY NAME AND ADDRESS	13. TYPE OF REPORT AND PERIOD COVERED
Research Office Washington State Department of Transportation Transportation Building, MS 47370 Olympia, Washington 98504-7370	Research report
	14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

This study was conducted in cooperation with the U.S. Department of Transportation, Federal Highway Administration.

16. ABSTRACT

A Fuzzy Logic Ramp Metering Algorithm will address the needs of Seattle's freeway system and overcome limitations of the existing ramp metering algorithm. This project progressed toward implementing and testing a fuzzy neural ramp metering algorithm on-line at the Traffic Systems Management Center (TSMC) for the Washington State Department of Transportation's Northwest Region. Improvements were made to neural network predictors to allow better generalization.

Code was written for the fuzzy ramp metering algorithm and its interface with the pre-existing TSMC code. Of the new code written, approximately 95 percent of it was for the interface, and only 5 percent of it was for the ramp metering algorithm itself. Interfacing the fuzzy controller with the existing TSMC software required modification of 16 pre-existing files related to the ramp metering database, real-time skeleton, and ramp metering and data collector communications.

A method was developed and code was written to directly send metering rates from the VAX computer to the 170 computer and to implement them, whereas previously only a metering rate adjustment had been possible. The operator interface was designed and code was written to enter fuzzy tuning parameters and fuzzy equations. The specifications for each new parameter were designed.

Although this code was written, it has not yet been implemented on-line because of time constraints. Preparation for on-line implementation required more time than anticipated because of the unexpected complexity of the pre-existing TSMC code. On-line implementation and testing will proceed on a WSDOT/TransNow project that begins in September 1997.

In addition to software design, further planning was necessary to ensure smooth implementation and quality performance. The testing plan was developed in greater detail to include software quality testing. Primary and backup study sites were chosen, and an evaluation technique was selected. A risk assessment plan was developed to mitigate future problems.

17. KEY WORDS	18. DISTRIBUTION STATEMENT
Artificial neural networks (ANN), fuzzy logic controller (FLC), traffic data prediction, ramp metering	No restrictions. This document is available to the public through the National Technical Information Service, Springfield, VA 22616

19. SECURITY CLASSIF. (of this report)	20. SECURITY CLASSIF. (of this page)	21. NO. OF PAGES	22. PRICE
None	None	60	

Research Report
Research Project Agreement No. T9903, Task 70
Fuzzy Neural Seattle

ON-LINE IMPLEMENTATION OF A FUZZY NEURAL RAMP METERING ALGORITHM

by

Cynthia E. Taylor
Research Engineer

Deirdre R. Meldrum
Associate Professor

Department of Electrical Engineering
University of Washington, Box 352500
Seattle, Washington 98195

Washington State Transportation Center (TRAC)
University of Washington, Box 354802
University District Building
1107 NE 45th Street, Suite 535
Seattle, Washington 98105-4631

Washington State Department of Transportation
Technical Monitor
Les Jacobson
Traffic Systems Manager, Northwest Region

Sponsored by

Washington State Transportation Commission
Department of Transportation
Olympia, Washington 98504-7370

Transportation Northwest (TransNow)
University of Washington
135 More Hall, Box 354802
Seattle, Washington 98195

and in cooperation with
U.S. Department of Transportation
Federal Highway Administration

August 1997

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. This document is disseminated through the Transportation Northwest (TransNow) Regional Center under the sponsorship of the U.S. Department of Transportation UTC Grant Program and through the Washington State Department of Transportation. The U.S. Government assumes no liability for the contents or use thereof. Sponsorship for the local match portion of the research project was provided by the Washington State Department of Transportation. The contents do not necessarily reflect the official views or policies of the U.S. Department of Transportation, Washington State Transportation Commission, or Washington State Department of Transportation. This report does not constitute a standard, specification, or regulation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

This research project made progress toward implementing and testing a fuzzy neural ramp metering algorithm on-line at the Traffic Systems Management Center (TSMC) for the Northwest District. Improvements were made to neural network predictors to allow better generalization.

Code was written for the fuzzy ramp metering algorithm and its interface with the pre-existing TSMC code. Of the new code written, approximately 95 percent of it was for the interface, and only 5 percent of it was for the ramp metering algorithm itself. Interfacing the fuzzy controller with the existing TSMC software required modification of 16 pre-existing files related to the ramp metering database, real-time skeleton, and ramp metering and data collector communications.

A method was developed and code was written to directly send metering rates from the VAX computer to the 170 computer and to implement them, whereas previously only a metering rate adjustment had been possible. The operator interface was designed and code was written to enter fuzzy tuning parameters and fuzzy equations. The specifications for each new parameter were designed.

Although this code was written, it has not yet been implemented on-line because of time constraints. Preparation for on-line implementation required more time than anticipated because of the unexpected complexity of the pre-existing TSMC code. The task of studying the TSMC code that relates to the ramp metering interface required eight more months than had been planned. On-line implementation and testing will proceed on a WSDOT/TransNow project that begins in September 1997.

In addition to software design, further planning was necessary to ensure smooth implementation and quality performance. The testing plan was developed in greater detail to include software quality testing. Primary and backup study sites were chosen, and an

INTRODUCTION

On-line studies have shown that ramp metering effectively increases freeway efficiency at a relatively low implementation cost (Robinson and Doctor, 1989). Even a slight improvement in ramp metering performance may reduce accident rates and produce significant monetary savings. Most ramp metering algorithms could use improvement in their response time, robustness to errors in sensor data, and performance under non-recurrent congestion. Although the bottleneck ramp metering algorithm used in the Seattle area is one of the most sophisticated in the country, it has limitations. The objective of this research was on-line implementation and testing of a fuzzy neural ramp metering algorithm for the Seattle area to improve freeway efficiency.

The development, simulation testing, and on-line implementation of the fuzzy neural ramp metering algorithm has been an ongoing project. The algorithm was developed under a WSDOT project (Taylor, 1994; Meldrum and Taylor, 1995) to overcome the limitations of Seattle's bottleneck algorithm. This algorithm was also tested in simulation under a TransNow grant (Taylor and Meldrum, 1995). A multiple ramp study site from the Seattle I-5 corridor was modeled with the freeway simulation software FRESIM. In five of the six tests encompassing a variety of traffic conditions, the fuzzy controller outperformed the three other controllers tested in terms of total distance traveled, total travel time, and vehicle delay.

Two other research groups have demonstrated that fuzzy logic control is appropriate for ramp metering applications. A CALTRANS group tested a fuzzy logic controller (FLC) to control entry to the San Francisco-Oakland Bay Bridge (Chen and May, 1990). Their ramp metering algorithm performed well in simulation and has been implemented on-line. In Holland, an FLC has been tested for on-line ramp metering on the A12 freeway between The Hague and Utrecht (Taale, Slager, and Rosloot, 1996). The algorithm restricts the metering rate when the downstream speed is lower than the upstream speed. The FLC

produced 35 percent faster travel times and a 5 to 6 percent greater bottleneck capacity than two other controllers for an 11-kilometer freeway section. Our fuzzy ramp metering algorithm differs from these algorithms because it controls multiple ramps, uses more comprehensive inputs, and uses different heuristics.

RESEARCH APPROACH

This research project was intended to finish implementation and testing of the fuzzy ramp metering algorithm on the Seattle freeway system. However, research tasks took much longer than expected, and additional unanticipated research tasks were necessary for successful implementation.

The major stumbling block of this project was the study of the TSMC's existing software. This study was complicated by several factors. One, there was a lack of documentation regarding program structure and function. The TSMC programming had been contracted out to Howard Needles Tammen & Bergendoff (HNTB), but the contract did not provide software support or modification. Consequently, no one at WSDOT had particular knowledge of the TSMC software. Two, the TSMC code was much more complex and convoluted than expected, consisting of 2891 files in 61 directories. Unfortunately, the portions of code that interface with ramp metering were not isolated. Interfacing the fuzzy controller with the existing TSMC software required modification of sixteen pre-existing files related to the ramp metering database, real-time skeleton, and ramp metering and data collector communications. In fact, most of the new code was written for the interface between the ramp metering algorithm and the pre-existing TSMC software, rather than for the algorithm itself. Historically, software is developed rapidly and somewhat haphazardly, then debugged on-line. However, when the software affects a safety critical operation such as the TSMC, development requires more careful planning to anticipate any possible ramifications of software modification and to eliminate any bugs before the software is brought on-line.

Several additional research tasks were completed to ensure a quality product, although the time line had not anticipated these tasks:

1) Provide documentation of the pre-existing TSMC software that interfaces with ramp metering.

2) Design and write code for the operator interface to enter fuzzy equations and tune parameters.

3) Develop new VAX-170 communications protocol for directly implementing metering rates.

4) Define specifications for all new elements in the database.

5) Design a software testing plan.

6) Refine the on-line testing plan.

7) Choose primary and backup testing sites.

8) Determine an evaluation technique.

9) Assess risks and develop a plan to mitigate them.

This report documents the findings from these tasks.

FINDINGS

The results from this research project include neural network improvements, TSMC software documentation, new software for implementation of the fuzzy ramp metering algorithm, design of the operator interface, specifications and tuning of the new database elements, and instructions for entering the fuzzy equations.

NEURAL NETWORK IMPROVEMENTS

Neural networks to predict volume and occupancy 1 minute in advance were designed in a previous WSDOT research project (Taylor 1994; Meldrum and Taylor, 1995). The objective was to use the predictions as inputs to the fuzzy ramp metering algorithm to create a pro-active controller. However, although the neural networks could predict well for the data set on which it was optimally tested, it did not generalize well to a third set used for cross-validation. Because the neural networks were not accurately generalizing to new data, improvements were made to overcome this problem. The researchers experimented with several different architectures and training techniques.

The training and testing data sets were increased from 200 to 800 input/output examples. The original data sets had used data samples from three hours of one day. The newer data sets used samples from three hours of four days to train and test the network over a greater range of conditions. The larger training and testing set provided a better evaluation of the prediction performance.

Additional upstream data stations were added as inputs to the neural network. Previously, data from two stations had been used as inputs, but later data from four stations were used.

The data scaling technique was improved so that more of the data fit within the 0 to 1 range of the neural network output. The neural network had experienced significant saturation, meaning that neuron outputs would be stuck on an output of 0 or 1. With the

improved data scaling technique, the neural networks generalized somewhat better by saturating less frequently.

The amount of training needed for the network was determined optimally by plotting the training and testing performance versus iterations to track the network's progress. This method saved time because each architecture only needed to be trained once. Optimal training revealed that the neural network generalized more poorly when it was overtrained. In fact, the neural network did best when it learned only the general trend of the training data. Although the network was capable of learning each data point almost perfectly, it could not generalize to the testing data when it was overtrained.

The architecture of the network, such as the number of hidden neurons and the number of past samples used in the network input, was varied. The network architecture did not appear to be as crucial as the network training technique.

The training procedure was modified to randomly select training examples rather than cycle through them sequentially. With random training, the testing performance fluctuated more as it learned, which allowed the network to avoid local minimums and learn more quickly. Consequently, randomized training produced better generalization than sequential training.

The training technique was altered to incorporate weight decay to limit large weight growth. Weight decay penalized large weights, encouraging the network to utilize more neurons. Weight decay helped prevent the testing error from escalating as the training iterations increased. Although the training error did not decrease beyond a certain point when weight decay was used, the generalization ability was improved.

One problem with the training technique involved differences between the training and testing set. Optimal testing performance would occur before the neural network had learned the training set very well. The error of the training data would still have a mean offset. An explanation for the poorly learned training set is that the training and testing data sets had different properties. The data sets were modified to ensure that they had the same

statistical properties by randomly scrambling them together. The result was that the training and testing error were nearly the same, as they should be for optimal training.

A third data set was added for cross-validation. The first data set was used to train the neural network. The second data set was used to determine when the neural network had been trained optimally. The third set validated the generalization ability of the network. No training occurred on the second and third set, only testing. Because of scrambling, visual examination of network outputs could no longer be used to evaluate the network performance. However, the validation set helped overcome this problem because it had not been scrambled, allowing visual examination.

Figure 1 shows occupancy prediction results on a representative portion of the validation set. The solid line shows the predicted occupancy, and the dotted line shows the actual occupancy. The predicted occupancy captures the correct data trend, but it does not always reach the extreme highs and lows. This neural network has two hidden neurons. The inputs include the three past samples of occupancy, as well as occupancy for four current stations. Optimal performance on the testing set occurs at 4000 iterations over the training set, where the mean squared error of the training set is 2.92, and the mean squared error of the testing set is 2.84. The mean squared error of the validation set shown is 2.25.

Despite these improvements to the neural networks, the prediction performance was still unsatisfactory when the networks generalized to new data. In addition, training the neural networks for each predictor site proved to be an arduous task. Because the inputs to the fuzzy controller encompassed those of the neural network, there was some redundancy in including the neural networks. In simulation, the previously sampled value of upstream occupancy provided a satisfactory substitute for the predictive input. By increasing the weighting factor of the rules that check for extremes in upstream and downstream traffic data, these rules served the same function as the predictive inputs and performed well in simulation (Taylor and Meldrum, 1995). In addition, the simpler the controller, the easier it is to tune and maintain. With these factors in mind, the researchers decided that the

neural networks were not worth the considerable additional complexity they added within the fuzzy ramp metering algorithm. Because of time and financial constraints, it was not feasible to train neural network predictors for the entire ramp metering system. The fuzzy ramp metering algorithm will initially be implemented and tested without using the predictive input from the neural networks. If time and resources allow, the neural network predictors will be added later.

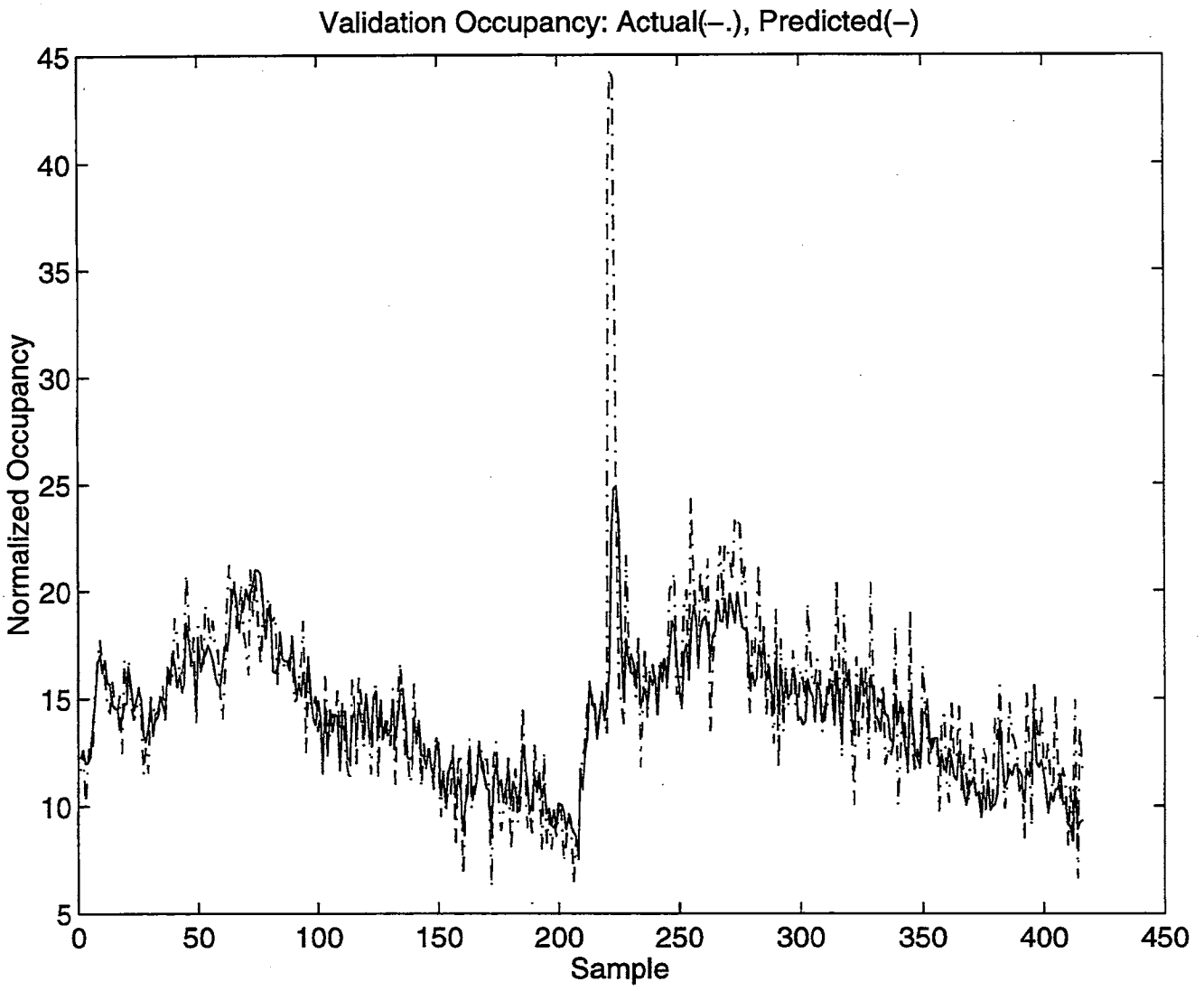

Figure 1: Neural Network Prediction Results

TSMC SOFTWARE DOCUMENTATION

The portions of code that related to the ramp metering algorithm were studied to understand the effects of any modifications. Familiarity with the pre-existing software was necessary to design the interface for the ramp metering algorithm. The portions of code that were studied were documented for reference. This documentation is supplemental to that provided by HNTB and may help programmers who modifiy the code. The documentation, titled "Documentation of TSMC Software that Interfaces with Traffic Analysis Programs," is divided into five major sections:

1) code structure
2) Vax-170 communications
3) real-time software
4) traffic analysis programs
5) databases.

Code Structure

Section 1 of the TSMC documentation contains the directory structure, a tags file, and calling trees. The tags file was created with the UNIX command **ctags**. The tags file alphabetically lists the name of each module in the TMS code, its location, and the search string used to identify it. The tags file is a powerful tool for locating modules, a valuable feature for large systems such as the TSMC. The vi or emacs editor (and possibly VMS edit) can utilize the tags file to go to any module. For example,

vi -t *module_name*

finds the file containing the module_name and opens it for editing. From vi,

:ta *module_name*

hops down to the file containing the module_name and opens it for editing.

:po

pops back up to the file containing the previous module. In this manner, tags allow the user to go up or down any number of levels. The tags file can be found on the TSMC Vax in DUA2:[taylorc.code]code.tags.

The calling trees were created with the UNIX tool **calls**. Our documentation contains calling trees for the mains, of which there are about 60. The path name relative to the main location is given in brackets following each module name (the syntax is in UNIX, where .. means up a directory and / means down a directory). The calls trees were edited to include only the main and the modules called by it. It was impractical to document call trees for every module, of which there are over 1000. Sometimes the calling trees were lacking in detail in some places where **calls** had trouble reading the code, but they did provide a good starting point in understanding how the modules interact.

Vax-170 Communications

Section 2 of the TSMC documentation clarifies when each command is issued and what it contains. It describes the Ramp Meter/Data Collectors Communications Handler (RMDC_COMM). The RMDC_COMM processing loop performs several functions. It checks for TMS shutdown, processes keyboard interrupts, processes mailbox interrupts, processes data received from the 170s, processes buffers ready to send to the 170s, and sends data polls to the 170s. For implementing parameter changes and metering rates for the fuzzy algorithm, the main area of interest here the RMDC is updated. When an operator makes a change from the operator console, or **patch_rmdb**, the changes are sent through mailbox interrupts after the update flag has been set.

Real-Time Software

The documentation in Section 3 of the TSMC documentation describes **tms_startup**, **tms_shutdown**, the real-time skeleton, and error handling. **Tms_startup** is of interest because it starts up the traffic analysis programs, including fuzzy ramp metering. When **tms_shutdown** is executed, it sets shut-down flags that are checked by the running processes. Fuzzy ramp metering uses the same shutdown flag as the other real-time processes, so it does not require any changes to the shutdown process. The real-time skeleton executes a loop every 20 seconds, which starts up polling processes (**multi_rmdc_comm**, **test_rmdc_comm**, and **dummy_data**), starts up traffic

analysis programs (including fuzzy metering), and then scrolls the real-time database, at which point the database is stable. Next, it runs the communications handlers and monitoring programs. Several modules handle different types of errors. Fuzzy metering uses the same error handling functions as the other Traffic Analysis Programs (TAPS). Errors are written either to the terminal, the fddb error file, the TMS Event Logger, or the Communications Handler Log File.

Traffic Analysis Programs

Section 4 of the TSMC documentation records the TAPS, which include bottleneck, incident detection, and station aggregation. Fuzzy metering is integrated as one of the TAPS, so it is particularly relevant how these processes coordinate their timing, obtain parameters from the Ramp Metering Data Base (RMDB), obtain real-time data, and implement their results. Upon startup, the TAPS build a table of pointers in global memory that allow quick access to parameters in the RMDB and data in the Real-Time Database (RTDB). Every twenty seconds, a flag set by the real-time skeleton indicates that it is time to execute the real-time processes.

Databases

Build_rmdb is by far the most complex of the software studied because it starts a long chain of functions that call other functions with insufficient software documentation. **Build_rmdb** opens and parses "rmdb_input.fil", builds RMDB, and creates temporary files that are later used to build tables in global memory for TAPS. It also sorts names, loops, stations, and speed traps before writing them to "rtfmdbname.srt" to be used for later creation of RTDB and the Five-Minute Database (FMDB). **Build_rmdb** needed modification to include the fuzzy parameter and fuzzy equations. The documentation in Section 5 of the TSMC documentation explains the levels of defaults, structure of the RMDB, and operation of **build_rmdb**. The structure and building of the RTDB are described there as well.

NEW SOFTWARE

New code was written for the fuzzy ramp metering algorithm to operate on the TSMC VAX. Table 1 lists the new modules and summarizes the function that they perform. Appendix A, Section 1, contains detailed pseudo code and description of the new **fuzzymeter** modules. Most of the new code is for interfacing **fuzzymeter** rather than for the algorithm itself. To interface **fuzzymeter** with the existing TSMC software, changes were necessary to **build_rmdb**, the RMDB structure, and **rmdc_comm**. Table 2 lists the pre-existing files that were modified and summarizes the changes made to them. Several changes were necessary to functions that are called from **build_rmdb** to process the fuzzy parameters and fuzzy equations (Appendix A, Section 2). The structure of the RMDB was modified to include the new group names and new parameter names (Appendix A, Section 3). Changes were necessary to the **rmdc_comm** files to directly implement a metering rate in the 170s. Two options were available for directly implementing the rate: 1) set the minimum and maximum allowable metering rate to be equal to the desired metering rate, and 2) modify the VAX-170 communications protocol to send the metering rate directly. The required changes for both of these methods were analyzed (Appendix A, Section 4). The second method required changes not only to **rmdc_comm** but also to the 170 logic. The second method was recommended not only because the first method was nontrivial but because the second method was desirable for proper, final implementation.

OPERATOR INTERFACE

The operator interface with the fuzzy ramp metering algorithm was designed, and code was written for it. The way in which **fuzzymeter** interfaces to the TSMC software is exactly parallel to the other TAPS. The "**rmdb_input.fil**" is used to build the RMDB off-line. It allows the operators to enter fuzzy parameters that differ from the defaults and to enter fuzzy equations that specify the loops that are used as inputs to the controller.

Table 1. New Modules

Module Name	Description of Function
fuzzymeter	Main process that maps to global memory, starts up processes, and activates **fuzzymeter** calculation when flagged by the real time skeleton
build_fuzzymeter_table	Module called from **fuzzymeter**. Parses the **fuzzymeter** equation file that was created by **build_rmdb**. Searches for cabinet names in RMDB and station names in RTDB. Writes action codes and data pointers to a table of pointers to allow quick access to the RMDB and to the RTDB
calc_fuzzymeter	Module called from **fuzzymeter**. Obtain inputs to the fuzzy controller. Parses fuzzy meter table. Fetches data from the RTDB and fuzzy parameters from the RMDB. Calls **calc_fuzzy_rate** and implements resulting metering rate.
calc_fuzzy_rate	Module called from **calc_fuzzymeter** to calculate a metering rate given inputs to the fuzzy controller. This is where the fuzzy ramp metering algorithm begins.
fuzzify	Module called from **calc_fuzzy_rate**. Converts numerical input to a set of fuzzy classes
rules	Module called from **calc_fuzzy_rate** to evaluate the rule base
defuzzify	Module called from **calc_fuzzy_rate**. Converts fuzzy metering classes to a numerical metering rate
watch_fuzzymeter	Main process that observes **fuzzymeter** and displays metering rates produced by the controller
cl_fuzzy_meter.com	Command file to compile and link **fuzzymeter**
cl_watch_fuzzymeter.com	Command file to compile and link **watch_fuzzymeter**

Table 2. Modified Modules

Module Name	Description of Function
tms_startup.c	Add call to **start_tms_process** to start **fuzzymeter**
build_rmdb.c	Add capability to parse and process fuzzy parameters and fuzzy equations from "rmdb_input.fil"
tap.h	Define new action codes used in the **fuzzymeter** table
event_common.h	Include facility numbers for new TMS software modules
rmdb_func_prot.h	Include function prototype for **get_fuzzy_eqn**
rmdc_comm.c	Change to use new data polling module and to initiate new parameters when there is a communications failure
rmdb_comm_sub.c	Add new function named **build_and_queue_170_meter_rate** and a call to this function. This function builds a new type of data poll that contains direct metering rates for lanes 1, 2, and 3
rmdc_comm.h	Change to use new data polling module. Define new request for data poll that sends metering rate directly.
rt_skeleton.c	Add call to **fuzzymeter** using **run_process_wait** from a control loop every 20 seconds
rmdb_sub.c	Add new functions **get_fuzzy_eqn** and **get_next_fuz_line**
rmdb.h	Add new RMDB data structures and definitions for new RMDB group names arrays of structure rm_dc_data_column
rmdb_tbl.c	Add initial values for new parameters in minimum, maximum, default, and ep_mask data column. Add new parameters to the name_table of structure fddb_name_table. Add new groups and new parameters to output_list array of structure fddb_output_list. Add new group names to group_table array of structure fddb_group_table.
tms_realtime.h	Add new **fuzzymeter** event flags
copy_code_to_backup.com	Add new modules for backup
copy_realtime_to_test.com	Add new modules
rmdb_input.fil	Add new fuzzy equations and fuzzy parameters for metered ramps

From the operator console, the operator can control the fuzzy ramp metering algorithm, such as modifying parameters and switching between **fuzzymeter** and the bottleneck metering. **Patch_rmdb** can be used to make parameter changes directly from the VAX rather than from the PC. **Watch_rmdc** can be used to observe requested real-time data, such as inputs to the ramp metering algorithm. **Watch_fuzzymeter** will be used to observe the inputs and outputs of the fuzzy controller. The output files to which **fuzzymeter** writes include the operator log, journal file, TMS event logger, and Communications Handler Log File. For details of how these modules interact, see Appendix B.

DATABASE ELEMENTS

The parameters needed to operate and tune the fuzzy ramp metering algorithm were determined. Table 3 contains the parameter names, descriptions, codings, units, minimum value, maximum value, default value, and example input. Parameter names are structured into components combined in the following order where they are applicable: mainline location of the variable (Local/Loc, Down or Up), the variable type (AdvQueue, Queue, Occ, Speed/Sp, or MeterRate/Mr), the fuzzy class (Vs, S, M, B, Vb), the function of this parameter (Low, High, Wt), and the lane number of the metered ramp (1, 2, or 3). Because each on-ramp lane may have different vehicle sources (such as a left and right turn from the arterial), each ramp has independent control parameters. Hence, a fuzzy equation must be specified for each metered lane of the on-ramp. The function *Wt* sets the weighting factor for a particular rule, where "-" denotes the logical AND between two inputs of a rule premise. All of the new parameters defined in Table 3 are tunable. Two types of parameters affect the behavior of the controller in different ways: dynamic range limits (Low and High) and rule weights (Wts). The range limits affect the sensitivity range for a particular input, and the rule weights stress the relative importance of a particular rule. This section introduces the dynamic range limits, rule weights, and general tuning guidelines.

Table 3: Definitions of New Database Elements

NAME	DESCRIPTION	CODING	UNIT	MIN	MAX	DEFAULT	EXAMPLE
MeterRateLane1	Metering rate produced by fuzzy controller, lane 1	UBYTE1	VPM	0.0	25.5	0.0	MeterRateLane1 = 0.0
MeterRateLane2	Metering rate produced by fuzzy controller, lane 2	UBYTE1	VPM	0.0	25.5	0.0	MeterRateLane2 = 0.0
MeterRateLane3	Metering rate produced by fuzzy controller, lane 3	UBYTE1	VPM	0.0	25.5	0.0	MeterRateLane3 = 0.0
AdvQueueOccHigh1	High end of dynamic range for Advance Queue Occ, lane 1	USHORT1P	%	0.0%	100.0%	30.0%	AdvQueueOccHigh1 = 30.0
AdvQueueOccHigh2	High end of dynamic range for Advance Queue Occ, lane 2	USHORT1P	%	0.0%	100.0%	30.0%	AdvQueueOccHigh2 = 30.0
AdvQueueOccHigh3	High end of dynamic range for Advance Queue Occ, lane 3	USHORT1P	%	0.0%	100.0%	30.0%	AdvQueueOccHigh3 = 30.0
AdvQueueOccLow1	Low end of dynamic range for Advance Queue Occ, lane 1	USHORT1P	%	0.0%	100.0%	12.0%	AdvQueueOccLow1 = 12.0
AdvQueueOccLow2	Low end of dynamic range for Advance Queue Occ, lane 2	USHORT1P	%	0.0%	100.0%	12.0%	AdvQueueOccLow2 = 12.0
AdvQueueOccLow3	Low end of dynamic range for Advance Queue Occ, lane 3	USHORT1P	%	0.0%	100.0%	12.0%	AdvQueueOccLow3 = 12.0
AdvQueueOccWt1	Weight for Adv Queue Occupancy Rule, lane 1	UBYTE1	N/A	0.0	25.5	3.0	AdvQueueOccWt1 = 3.0
AdvQueueOccWt2	Weight for Adv Queue Occupancy Rule, lane 2	UBYTE1	N/A	0.0	25.5	3.0	AdvQueueOccWt2 = 3.0
AdvQueueOccWt3	Weight for Adv Queue Occupancy Rule, lane 3	UBYTE1	N/A	0.0	25.5	3.0	AdvQueueOccWt3 = 3.0
DownOccHigh	High end of dynamic range for Downstream Occupancy	USHORT1P	%	0.0%	100.0%	18.0%	DownOccHigh = 18.0
DownOccLow	Low end of dynamic range for Downstream Occupancy	USHORT1P	%	0.0%	100.0%	8.0%	DownOccLow = 8.0
DownSpeedHigh	High end of dynamic range for Downstream Speed	USHORT1	MPH	0.0	6553.5	60.0	DownSpeedHigh = 60.0
DownSpeedLow	Low end of dynamic range for Downstream Speed	USHORT1	MPH	0.0	6553.5	30.0	DownSpeedLow = 30.0
DownSpVs-OccVbWt	Weight for Very Small Speed and Very Big Occ Rule	UBYTE1	N/A	0.0	25.5	2.5	DownSpVs-OccVbWt= 2.5
DownSpS-OccBWt	Weight for Downstream Small Speed and Big Occ Rule	UBYTE1	N/A	0.0	25.5	2.5	DownSpS-OccBWt = 2.5
DownSpM-OccMWt	Weight for Downstream Medium Speed and Medium Occ Rule	UBYTE1	N/A	0.0	25.5	1.0	DownSpM-OccMWt = 1.0
LocalOccHigh	High end of dynamic range for Local Occupancy	USHORT1P	%	0.0%	100.0%	18.0%	LocalOccHigh = 18.0
LocalOccLow	Low end of dynamic range for Local Occupancy	USHORT1P	%	0.0%	100.0%	8.0%	LocalOccLow = 8.0
LocalOccVbWt	Weight for Local Very Big Occupancy Rule	UBYTE1	N/A	0.0	25.5	2.0	LocalOccVbWt =2.0
LocalOccBWt	Weight for Local Big Occupancy Rule	UBYTE1	N/A	0.0	25.5	1.0	LocalOccBWt = 1.0
LocalOccMWt	Weight for Local Medium Occupancy Rule	UBYTE1	N/A	0.0	25.5	1.0	LocalOccMWt = 1.0
LocalOccSWt	Weight for Local Small Occupancy Rule	UBYTE1	N/A	0.0	25.5	1.0	LocalOccSWt = 1.0
LocalOccVsWt	Weight for Local Very Small Occupancy Rule	UBYTE1	N/A	0.0	25.5	2.0	LocalOccVsWt = 2.0
LocalSpeedHigh	High end of dynamic range for Local Speed	USHORT1	MPH	0.0	6553.5	60.0	LocalSpeedHigh = 60.0
LocalSpeedLow	Low end of dynamic range for Local Speed	USHORT1	MPH	0.0	6553.5	30.0	LocalSpeedLow = 30.0
LocSpVs-OccVbWt	Weight for Local Very Small Speed and Very Big Occ Rule	UBYTE1	N/A	0.0	25.5	1.0	LocSpVs-OccVbWt = 1.0
LocalSpeedSWt	Weight for Local Small Speed Rule	UBYTE1	N/A	0.0	25.5	1.0	LocalSpeedSWt = 1.0
LocalSpeedBWt	Weight for Local Big Speed	UBYTE1	N/A	0.0	25.5	1.0	LocalSpeedBWt = 1.0
LocSpVb-OccVsWt	Weight for Local Very Big Speed and Very Small Occ Rule	UBYTE1	N/A	0.0	25.5	1.0	LocSpVb-OccVsWt = 1.0
MeterRateHigh1	High limit for metering rate produced by fuzzy controller, lane 1	UBYTE1	VPM	0.0	25.5	15.0	MeterRateHigh1 = 15.0
MeterRateHigh2	High limit for metering rate produced by fuzzy controller, lane 2	UBYTE1	VPM	0.0	25.5	15.0	MeterRateHigh2 = 15.0
MeterRateHigh3	High limit for metering rate produced by fuzzy controller, lane 3	UBYTE1	VPM	0.0	25.5	15.0	MeterRateHigh3 = 15.0
MeterRateLow1	Low limit for metering rate produced by fuzzy controller, lane 1	UBYTE1	VPM	0.0	25.5	6.0	MeterRateLow1 = 6.0
MeterRateLow2	Low limit for metering rate produced by fuzzy controller, lane 2	UBYTE1	VPM	0.0	25.5	6.0	MeterRateLow2 = 6.0
MeterRateLow3	Low limit for metering rate produced by fuzzy controller, lane 3	UBYTE1	VPM	0.0	25.5	6.0	MeterRateLow3 = 6.0
PermitFuzzyMr1	Enable fuzzy control at this meter	YES_NO	FLAG	N/A	N/A	NO	PermitFuzzyMr1 = YES
PermitFuzzyMr2	Enable fuzzy control at this meter	YES_NO	FLAG	N/A	N/A	NO	PermitFuzzyMr2 = YES
PermitFuzzyMr3	Enable fuzzy control at this meter	YES_NO	FLAG	N/A	N/A	NO	PermitFuzzyMr3 = YES

Table 3: Definitions of New Database Elements

QueueOccHigh1	High end of dynamic range for Queue Occupancy, lane 1	USHORT1P	%	0.0%	100.0%	30.0%	QueueOccHigh1 = 30.0
QueueOccHigh2	High end of dynamic range for Queue Occupancy, lane 2	USHORT1P	%	0.0%	100.0%	30.0%	QueueOccHigh2 = 30.0
QueueOccHigh3	High end of dynamic range for Queue Occupancy, lane 3	USHORT1P	%	0.0%	100.0%	30.0%	QueueOccHigh3 = 30.0
QueueOccLow1	Low end of dynamic range for Queue Occupancy, lane 1	USHORT1P	%	0.0%	100.0%	12.0%	QueueOccLow1 = 12.0
QueueOccLow2	Low end of dynamic range for Queue Occupancy, lane 2	USHORT1P	%	0.0%	100.0%	12.0%	QueueOccLow2 = 12.0
QueueOccLow3	Low end of dynamic range for Queue Occupancy, lane 3	USHORT1P	%	0.0%	100.0%	12.0%	QueueOccLow3 = 12.0
QueueOccWt1	Weight for Queue Occupancy Rule, Lane 1	UBYTE1	N/A	0.0	25.5	2.0	QueueOccWt = 2.0
QueueOccWt2	Weight for Queue Occupancy Rule, Lane 2	UBYTE1	N/A	0.0	25.5	2.0	QueueOccWt = 2.0
QueueOccWt3	Weight for Queue Occupancy Rule, Lane 3	UBYTE1	N/A	0.0	25.5	2.0	QueueOccWt = 2.0
UpOccHigh	High end of dynamic range for Upstream Occupancy	USHORT1P	%	0.0%	100.0%	18.0%	UpOccHigh = 18.0
UpOccLow	Low end of dynamic range for Upstream Occupancy	USHORT1P	%	0.0%	100.0%	8.0%	UpOccLow = 8.0
UpOccMWt	Weight for Upstream Medium Occupancy Rule	UBYTE1	N/A	0.0	25.5	1.0	UpOccMWt = 1.0
UpOccSWt	Weight for Upstream Small Occupancy Rule	UBYTE1	N/A	0.0	25.5	2.0	UpOccSWt = 2.0
UpOccVsWt	Weight for Upstream Very Small Occupancy Rule	UBYTE1	N/A	0.0	25.5	2.0	UpOccVsWt = 2.0

More detailed tuning instructions will be developed from lessons learned during on-line testing.

Tuning the dynamic range limit

The Low and High parameter functions set the dynamic control range for that variable during the fuzzification process. The fuzzification process translates each controller input into five fuzzy classes: very small, small, medium, big, and very big (Vs, S, M, B, Vb), as shown in Figure 2 for local mainline occupancy. This process evaluates the degree to which each class is true on a scale of 0 to 1. In this example, LocalOccLow = 8.0 and LocalOccHigh = 18.0. Suppose that the occupancy input is 16.0. Then the big class is true to a degree of 0.6, and the very big class is true to a degree of 0.2. The remaining classes are 0 for this input value. If the occupancy input is less than the low limit of 8.0, the very small class is true to a degree of 1.0, and the remaining classes are 0. If the occupancy input is greater than the high limit of 18.0, then the very big class is true to a degree of 1.0. The remaining classes are 0. Thus, the controller is active even when the input is outside of the dynamic range limits, but the behavior is static rather than dynamic.

To adjust the dynamic control range to an input, the Low and High limits are modified, and the fuzzy classes will be proportionately rescaled between these limits. For example, suppose that the flow-density characteristics differ from the default for a particular section of highway so that transition from free-flow conditions to congestion occurs at a lower occupancy. By reducing the High limit of the occupancy from 18.0 to 17.0, the fuzzy classes will be redefined as shown in Figure 3. Now if the occupancy input is 16, the degree of the big class decreases from 0.6 to 0.24, and the degree of the very big class increases from 0.2 to 0.55. The consequences of this parameter change are apparent when the rules that use these premises are observed. The rule "If the local occupancy is very big, then the metering rate is very small" will fire to a greater degree. Meanwhile, the rule "If the local occupancy is big, then the metering rate is small" will fire

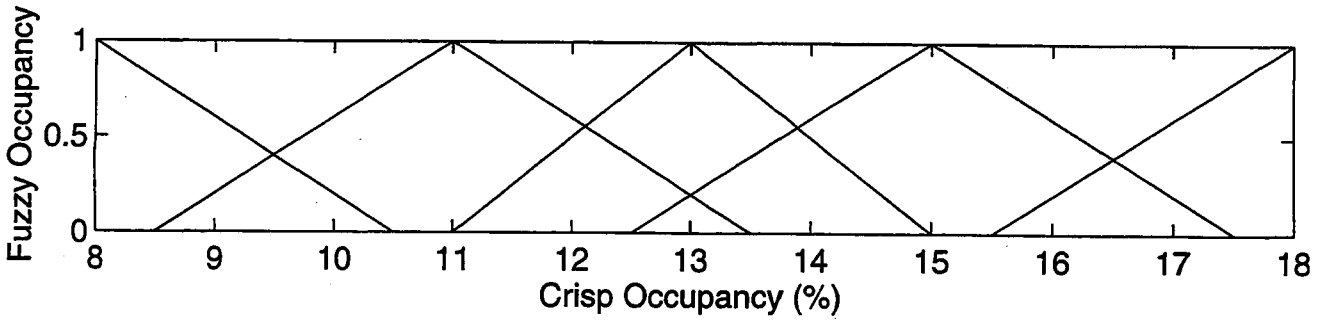

Figure 2: Fuzzy Classes for Occupancy where high limit is 18.0

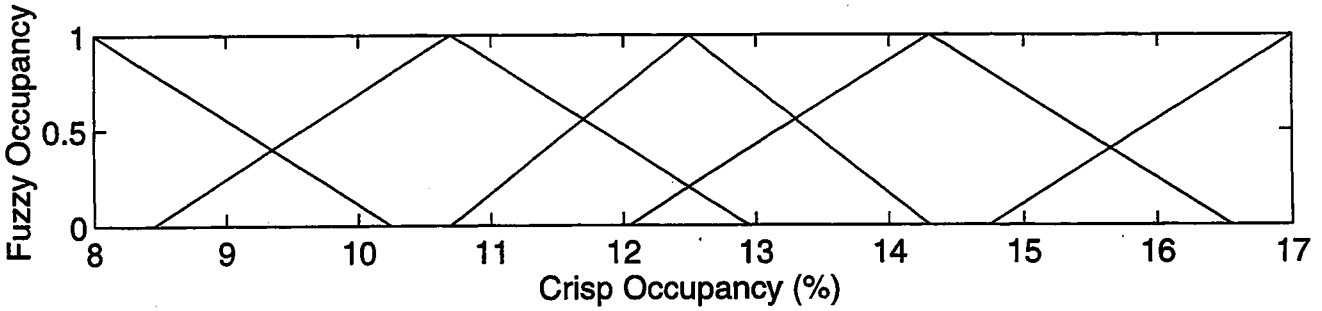

Figure 3: Fuzzy Classes for Occupancy where high limit is 17.0

to a lesser degree. If this is the only change, the resulting metering rate will be slightly more restrictive, given the same local occupancy.

For the queue occupancy and advance queue occupancy inputs, the fuzzy classes are defined differently so that only one of the five fuzzy classes and one corresponding rule are needed. Figure 4 shows how the very big class is stretched to begin rule activation at the Low limit of 12, with increasing activation up to the High limit of 30. If the queue occupancy is less than the Low limit, its fuzzy class degree is zero. If the queue occupancy is greater than the High limit, its fuzzy class is 1. Effectively, any time the queue occupancy is greater than the low limit, the rule that uses that input ("If the queue occupancy is very big, then the metering rate is very big") will fire to some degree.

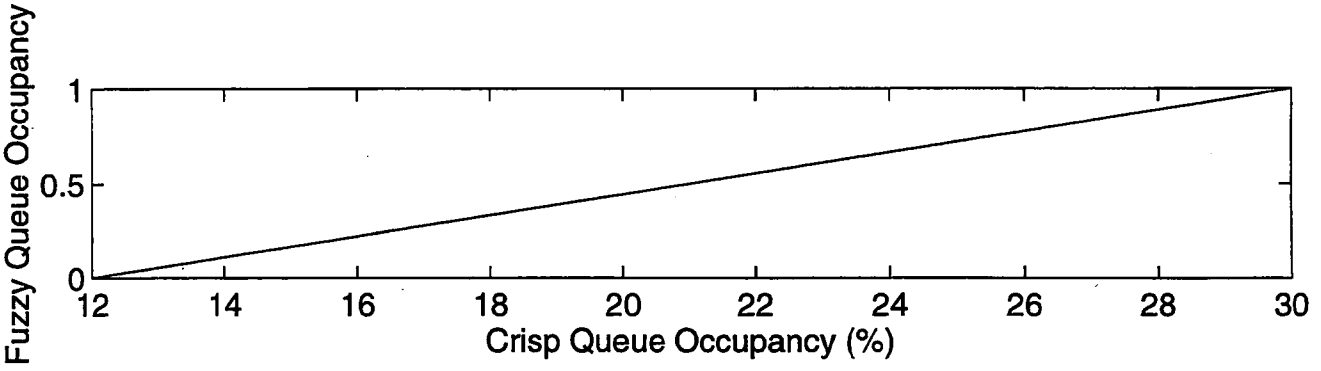

Figure 4: Fuzzy Class for Queue Occupancy

The limits of MeteringRateLow and MeteringRateHigh are used during the defuzzification process, the opposite of the fuzzification process. These parameters are the effective minimum and maximum metering rates that the fuzzy controller can produce. These defuzzification parameters are in addition to the minimum and maximum metering rates used system-wide. Both checks occur, so the fuzzy metering limits should be within the system-wide limits for a given metered lane.

Tuning the Rule Weights

For each rule, there is a corresponding parameter function to indicate the relative importance of that rule. These rule weights are used in the defuzzification process to produce a numerical metering rate given a set of fuzzy rule outcomes. A higher rule weight indicates that a rule is more important. For example, the default weighting of rule "If queue occupancy is very big, then the metering rate is very big" is 2, whereas most other rule weightings are 1. This means that the outcome of this rule will be weighted twice as heavily as most other rule outcomes. The weighting of the rule "If advanced queue occupancy is very big, then the metering rate is very big" is 3 because a large advance queue occupancy is more urgent than a large queue occupancy.

Tuning Guidelines

With tuning parameters for every input and rule, the controller can be tuned to produce a variety of behaviors. The advantage of this design is flexibility to meet a variety of situations. The disadvantage of this design is that improper tuning may produce unexpected results. The key to successful tuning is minor modification of a few parameters. All rule weights are relative to each other, so increasing one parameter may have the same effect as decreasing the remaining parameters. Obviously, it is better to modify as few parameters as necessary to understand the control effect.

There is a balance between the need to relieve mainline congestion with a restrictive metering rate and the need to relieve the ramp queue with a high metering rate. Most tuning can be done by modifying a couple of key parameters that affect this balance. The most commonly tuned parameter will be the rule weights for the queue occupancy and advance queue occupancy. They will be adjusted on the basis of factors specific to that location, such as how much storage space is available on the ramp, where the detectors are located, and political considerations. For example, if there is a greater than typical distance between the queue occupancy detector and the advance queue occupancy detector, the queue occupancy rule weight can be lowered slightly because there is more room for queue storage. If the advance queue detector is located on the arterial rather than at the ramp entrance, the weighting factor should be lower because *very big* occupancy on the arterial during a red signal phase does not necessarily indicate that the ramp itself is too full. On the other hand, a *very big* advance queue occupancy at the ramp entrance means that the ramp is fully occupied, and this rule should have a higher weighting factor. If demand on a particular ramp must be met for political reasons, the weighting factor for the queue occupancy and advanced queue occupancy should be increased.

Of the rules that pertain to the mainline, the ones that prevent bottleneck formation will most commonly be tuned. Suppose a downstream bottleneck commonly causes recurrent congestion at a particular location. To prevent bottleneck formation, the rule

weighting corresponding to that input should be increased: "If downstream speed is very small and downstream occupancy is very big, the metering rate is very small." If this bottleneck is typically one of the first to congest within a corridor, it should have a higher weighting factor for more preventative control.

ENTERING THE FUZZY EQUATIONS

For every metered lane that uses the fuzzy ramp metering algorithm, a fuzzy equation must be specified in the **rmdb_input.fil**. These equations are updated in the same way as the bottleneck equations; that is, the RMDB must be built off-line before **tms_startup**. The fuzzy equation must first specify the cabinet name and lane number to be metered, followed by an equal sign. Then the equation must specify the detector stations or loops to use for each input to the controller, as shown in the following prototype and example.

Prototype:

cab:loop = LOCAL | DOWN1 & DOWN2 & DOWN3 | UP1 & UP2 & UP3 | QUEUE & INT_QUEUE | ADV_QUEUE(X) & ADV_QUEUE(Y) & ADV_QUEUE(Z)

Example:

ES-158R:MMS_FM1 = ES-158R:MMS_Stn | ES-156R:MMS_Stn & ES-154R:MMS_Stn | ES-160R:MMS_Stn & ES-161R:MMS_Stn | ES-158R:MMSQ_1 & ES-158R:MMSI_1 | ES-158R:MMSRA_1(2) & ES-158R:MMSLA_1(4)

For all inputs, it is possible to use loops, stations, or a combination of both. (Both the occupancy and volume used to calculate speed are averaged per lane to standardize the units.) If multiple stations/loops are given for a location, they are averaged together (with error checking) to produce a single input. Typically, stations will be used for mainline inputs and loops will be used for ramp inputs. This section describes each controller input and rules for writing equations.

Five Detector Locations Used as Controller Inputs

1) The local adjacent mainline detector, typically located just upstream of the on-ramp, is used to gather local occupancy and local speed inputs from the previous sample (20-second data from the RTDB). Only one station or loop may be specified.

2) Downstream detectors are used to gather the downstream occupancy and downstream speed from the previous sample. Where possible, these detectors should be within bottleneck-prone sections to prevent heavy congestion. Up to five stations/loops may be specified. Use of more than one detector may be desirable if there are more than one distinct bottleneck downstream, or if the freeway flow splits into two major components (such as the I-90 exit off of I-5 south). Another situation in which to use more than one station is when the primary detector is prone to failure.

3) Upstream detectors gather upstream occupancy from the previous sample and correspond with rules that detect lighter congestion and approaching gaps between platoons. Up to five stations/loops may be specified. Use of more than one upstream station may be desirable when the freeway flow has two major sources.

4) The queue occupancy detector averages queue occupancy over the past five samples (100 seconds) and corresponds to rules that prevent excessive queue formation. Up to two stations/loops may be specified for this location. In the event that the queue occupancy detector fails frequently for a particular ramp, the intermediate queue occupancy detector may be specified as well. If both detectors are working, they will be averaged together. If one fails, the other will be used, maintaining operation capability. However, it is not necessary to specify the intermediate queue detector because the fuzzy logic controller checks for sufficient data. If input data are insufficient, operation will fall back to the bottleneck algorithm (if enabled) or to the local metering algorithm.

5) The advanced queue detector must be followed by the number of samples in parenthesis. The number of samples used to calculate this input is tunable because this detector location varies between the arterial and ramp. If the detector is located at the

entrance to the ramp, a small number of samples is recommended for faster response time to an excessive queue (approximately two samples). If the detector is located on the left hand turn for the arterial, a longer sample interval is recommended to represent the signal phase (approximately at least six samples for 120 seconds' duration or longer). If the detector is located on the right hand turn for the arterial, a shorter duration is recommended to capture recent fluctuations (approximately four samples for 80 seconds' duration). Up to five stations/loops are permitted at this location.

Rules for Writing Fuzzy Equations

- Lane # to meter must be given as the last character in the cabinet/loop name, as shown in the example equation on the previous page.
- Only one station/loop is allowed for the *Local* detector.
- Up to five stations/loops are allowed for *Downstream, Upstream,* and *Advance Queue* locations.
- Up to two Queue stations/loops are allowed (the second could be the intermediate queue).
- The number of past samples used to calculate the Advance Queue Occupancy must be given in parenthesis following the advance queue detector name, with no spaces between.
- "|" must delimit stations/loops of different locations.
- "&" must delimit stations/loops of the same location.
- Complete 15-character cabinet:loop names must be given to specify detectors (the parser does not assume the current cabinet, as in bottleneck equations).
- Spaces are optional between station names and are ignored by the parser.
- Do not put more than one equation per line (this differs from the HNTB software).
- An equation to be continued on the next line must end with a delimiter "=", "|", or "&".

APPLICATION

TESTING PLAN

The testing plan was developed in detail. A plan to evaluate software quality and algorithm quality was designed. Primary and backup study sites were chosen. A risk assessment plan was developed to avoid future problems.

Tests for Software Quality

The original proposal did not allow time for software testing. However, tests to verify that the new software operates properly are essential to ensure successful implementation. The following software tests were designed to thoroughly evaluate software quality.

- Fix any compile bugs.
- Fix any run bugs.
- Verify that **fuzzymeter** main starts up properly from **tms_startup** along with the other TAPS.
- Use test data to verify that global variables are mapping to correct locations in RMDB and RTDB.
- Verify that events and errors are logged to the correct file.
- Verify that **build_fuzzymeter_table** correctly builds a table of pointers based on test equations and test parameters in **rmdb_input.fil**.
- Verify that equations are parsed correctly. Incorrect equations will be given to verify that **build_fuzzymeter_table** catches erroneous inputs.
- Check that sufficient memory is allocated and reallocated for the table.
- Verify that **calc_fuzzymeter** starts up when an event flag signals from **rt_skeleton**.
- Verify that **fuzzymeter** terminates when an event flag signals shutdown for all tms.

- Verify that **calc_fuzzymeter** correctly retrieves parameters from the RMDB.
- Verify that levels of defaults work properly for RMDB.
- Using test data, check that data from the table are retrieved correctly.
- Using test data, check that data from RTDB are retrieved correctly.
- Using test data, check that **oper_permit_fuz** disable/enable works properly.
- Using test data, check that occupancy and volume are aggregated correctly.
- Using test data, check that speed is estimated correctly.
- Using test data, verify that the usability of the data is determined correctly.
- Verify that unusable data sets are logged and skipped.
- Using test data, check that correct parameters are passed to **calc_fuzzy_rate.**
- Verify that the metering rate range limits work properly.
- Verify that the metering rate converts correctly to an unsigned byte.
- Verify that the 170 uses the correct units of measurement for the metering rate.
- Using test data, verify that **calc_fuzzy_rate** works properly.
- Using test data for each class, verify that fuzzification works properly.
- Verify that the rule base is complete and that rule outcomes are as expected for test data.
- Verify that the defuzzifier produces the correct rate for test data.
- Time how long the TAPS take to execute. Verify that the maximum estimated run time is within an allowable range.
- Verify communications between the operator console and the VAX.
- Verify that the calculated metering rate is transmitted to and executed by the 170.
- Verify that the maximum estimated time for VAX-170 communications is within an allowable range.
- Make sure new parameters are updatable from the operator console.
- Verify that operators can switch between types of metering.
- Verify that RMDB changes for new parameters are written to the journal file.

- Verify that the operator console is displaying the correct parameter values.
- Verify that changes to the new parameters can be made from **patch_rmdb**.

Metrics for software quality

Because the TSMC performs vital functions that affect so many people, it is important that the software be reliable and easily maintainable. During software testing, the following metrics will be used to evaluate software quality:

- time economy of new real-time processes
- memory management
- sufficient functionality
- compatibility with existing software and ease of installation
- understandability of software
- ease of usability for operators
- security of software
- hardware independence from software
- expandability for future features
- error tolerance—the degree to which the software will continue to work without a system failure
- availability—the degree to which the software remains operable in the presence of system failures.

Tests for Algorithm Quality

On-line testing must be performed in a standardized method for valid results. The testing plan was designed for a uniform procedure, smooth implementation, thorough on-line evaluation, and proper test documentation.

On-line testing will use standardized procedures:

- Collect baseline corridor specifications and alternative route data.
- Follow general procedures for output data processing and analysis.
- Prepare hardware and software.

- Coordinate scheduling issues with TSMC personnel.
- Have a file naming convention and data storage plan to manage the files generated by testing.
- Consider safety issues.
- Clean up the equipment and software after testing.
- Document the following in a test log for each test performed:
    - date and time of each completed test
    - role of participants who conducted the test
    - hypothesis being tested and expected results
    - assumptions and constraints of test procedure
    - testing location description
    - testing conditions, prerequisite for start, duration of test, and time varying demand to capture the formation and dissipation of the rush hour
    - evaluation criteria and statistical methods used
    - input data sources and quality of data
    - actual results
    - conclusions.

Field testing of the ramp metering algorithm will take place in four phases in progressively more realistic operational environments. These phases are 1) off-line testing without data feedback, 2) off-line testing with hardware tests and no data feedback, 3) limited on-line field testing, with data feedback, and 4) extended on-line field testing with data feedback.

1) For the first phase of field testing, the ramp metering algorithm will be tested by being linked to the RTDB of WSDOT's freeway surveillance system. This linkage will be used to test the metering algorithms in the presence of real-world data to ensure that input data are being processed appropriately and that the computed metering rates are reasonable and as predicted. This test phase will be performed in an off-line environment without

affecting freeway operations; i.e., input data will be channeled to the metering algorithms from actual real-time data collected in the field, but algorithm outputs will be diverted to an off-line output file rather than to the operational freeway ramp meters. Although this limited testing environment cannot determine the effect of the generated metering rates, it will allow the rates to be compared to those produced simultaneously by the existing ramp metering algorithm used in the field to ensure that the new metering rate controller is behaving in a reasonable manner. Debugging of the metering algorithm will also be performed during this phase of field tests.

2) In the second phase of field testing, the metering algorithm test environment developed in the previous field testing phase will be supplemented with a field rack that mimics the metering hardware used in the field. This setup will then be used to vary the inputs to the ramp metering algorithm in an environment that closely resembles the field configuration, but still without affecting freeway operations. Every aspect of the metering controller will be tested one at a time to verify that

- the databases are built correctly
- the desired real-time data are being properly accessed
- the correct control parameters are being transmitted to the controller
- intermediate processing contains the correct information
- user inputs are being correctly interpreted by the controller
- the flags between the real-time skeleton and the controller are properly timed
- the inner calculations of the controller match manual calculations
- the generated metering rates are being transmitted to the field devices and executed properly.

By varying the demand of the field rack, this phase of field testing will allow further testing of the algorithm under variable conditions, as well as thorough testing of the software quality. As with the first test, there will be no output data feedback to determine the effect

of the generated metering rates, but comparisons can be made to the ramp metering algorithm that is concurrently operating.

3) Once off-line diagnostics have been completed, the algorithm will be tested on-line at the selected field site. The controller will interface with real-time inputs and field hardware for tuning the algorithm and verifying queue control functionality. These tests will also allow the study of the algorithm's computed metering rates in the presence of output feedback over time. Algorithm sensitivity to particular inputs in a real-time, on-line environment will also be adjusted during this phase of testing. Default parameter values were determined in simulation (Taylor and Meldrum, 1995) and will provide a good starting point. Further tuning will probably be needed on-line because of limitations in the simulation model. The metering algorithm will initially be installed at one ramp to verify that the algorithm performs appropriately in a real world environment at a single site before broader installation at other ramps in the test sites.

The new metering algorithm code is designed so that the new features can be disabled to allow the original metering algorithm software to operate as before. For example, when the new code is first put on-line, it will be put in a disabled mode to verify that the old portions of the software still work properly. This feature will help facilitate the implementation process and will ease the eventual transition from the existing metering rate algorithms to the new algorithm by allowing a smooth transfer from one algorithm to another algorithm under the full control of the operator.

4) In the final and most extensive field testing phase, comprehensive on-line field tests will be performed on the new metering algorithm at multiple metered ramps at two study sites. During this phase, activities will include monitoring of algorithm performance on the basis of internally generated performance measures; preliminary measurements of algorithm impacts on freeway operations; calibration and tuning of control parameters to further improve freeway efficiency; and development of default control parameters on the basis of a weighted cost function of performance measures. The algorithm tuning is an

interactive process; tests will have to be re-run to evaluate the changes against previous performance.

Because the metering rates of adjacent ramps affect each other, it is important to study the interaction of the ramp meters from a system context. For example, a more restrictive metering rate downstream may allow more generous metering rates upstream. The tests in this phase will explore the extent to which local optimization of metering rates typically produces global optimization, as well as the potential benefits of a hierarchical supervisory controller that explicitly performs global optimization on the basis of a macro-level viewpoint of several ramps.

This phase will evaluate the following aspects of the new ramp metering algorithm:

- performance during non-recurrent congestion, recurrent congestion, and transitional periods
- response to excessive ramp queue
- comparisons of metering rates and freeway performance prior to, during, and after the on-line installation period
- issues regarding installation, calibration, and maintenance
- lessons learned regarding on-line operation, performance monitoring, and future system-wide implementation tasks.

Because traffic is not uniform from day to day, testing will take place over a variety of background traffic conditions during a one-month period to determine the effect of individual algorithm adjustments. Among the factors to be included are weather conditions, special events, incidents, data quality, and seasonal variations. Explicit consideration of such conditions is important for two reasons: 1) it facilitates systematic testing over a broad range of conditions, and 2) it allows comparisons of algorithm performance on similar testing sets. Failure to take these outside factors into account may result in erroneous attribution of freeway performance changes to the metering algorithm, when in fact the variations in background conditions may be a significant causal factor.

Specific algorithm tuning tests will be isolated from one another during the testing process 1) to strengthen understanding of the causal relationships between the metering algorithm operations and freeway performance and 2) to develop background data regarding which specific tuning measures maximize the algorithm's efficiency. This process, while time consuming, strengthens the resulting analytical conclusions about the effectiveness of the metering rate algorithms, improves the algorithm's applicability to a broad range of conditions and locations, and enhances the prospects for successful future deployment.

During the data collection and analysis process, preliminary results will be reviewed to evaluate test plan progress and to determine the need for revisions to the test plan. In particular, phase four will be broken down into more specific tests as needed.

The results of all tests must be analyzed together to determine to what extent the real-time ramp metering algorithm meets the expected and desired system objectives. This analysis will reveal how the metering software performs under different conditions, what real or potential problems exist with meeting requirements, and what modifications are necessary to solve problems.

The test report will document the field tests and results:

- field test purpose
- field test procedure overview and equipment
- summary of background conditions, including weather, demand, and incidents
- summary of data types collected, units used, data processing technique, and statistical method
- analysis for various test scenarios
- ramp metering algorithm response and freeway operational performance during recurrent conditions, non-recurrent conditions, and transitional conditions
- ramp metering algorithm response and freeway operation performance during excessive queue demand
- coordination of ramp metering algorithm response to arterial signals

- comparison of metering rates prior to, during, and after the on-line installation period
- robustness over a broad range of conditions (variable weather, incidents, variable demand, missing detector data)
- transferability to different locations
- discussion of installation, calibration procedures, and maintenance issues
- procedure for on-line operations, algorithm tuning, and performance monitoring
- summary of on-line testing conclusions
- recommendations for future testing and implementation tasks.

Metrics for Algorithm Quality

The evaluation of the algorithm performance will be carried out according to the WSDOT freeway evaluation methodology for FLOW, developed in the "FLOW System Evaluation Framework" project. The same data processing techniques, units, and statistical methods will be used. The field testing sites encompass FLOW evaluation points for which we expect to have "before" data and, thus, will only need to collect "after" data for comparison. The following measures of performance will be used to evaluate algorithm quality:

- occupancy
- volume
- speeds
- travel times
- queue characteristics (size and delay)
- local and facility congestion patterns (CCTVs are useful here)
- incident patterns
- robustness over a broad range of conditions (recurrent and nonrecurrent congestion, variable weather, variable demand)
- versatility in different locations with varying geometry.

Experience with traffic control systems has taught two important lessons. First, no single parameter will generally serve as an effective measure under all traffic conditions. The system objectives must be defined to measure the system performance under different conditions. Second, the optimization measures used in the control algorithm must be measurable with the available surveillance tools.

The performance measures will be examined to determine the extent to which the new algorithm increases volumes while improving travel times and reducing delay. Of these measures, volume, occupancy, and speed can be measured directly. All others must be derived or estimated. Data analysis activities will be supported by FLOW evaluation tools, and the data analysis process will be documented in the testing results.

Test Sites

Four on-line study sites were chosen on the basis of input from Mark Morse, Mahrokh Arefi, and Greg Leege at WSDOT. The first two of these will be used for preliminary on-line testing, and the other two will be used as a backup (Table 4).

Table 4. Test Sites

Sight	Metered Ramps	Flow Sight
SB 405 AM	NE 124th St	
	NE 116th St	
	NE 85th St	<--
NB I-5 PM	NE 145th St	<--
	NE 175th St	
	County Line	
SB I-5 AM	Swamp Creek	
	44th Ave W	
	220th St SW	
WB I-90 AM	Issaquah Rd SE	
	W Lake Sammamish Way	
	Eastgate	<--

The following factors were considered when choosing these sites:

- a set of adjacent metered ramps
- the occurrence of recurrent congestion to provide relatively uniform test conditions for comparing different algorithms
- the occurrence of nonrecurrent congestion and special events to test under a broad range of conditions
- adequate surveillance already in-place in addition to loop detectors, CCTV, and speed sensors
- FLOW evaluation points to obtain "before" data
- in some cases, geographical isolation from the rest of the ramp metering system to allow the new algorithm to be evaluated independently from co-existing controllers
- no new construction planned for these sites.

RISK MANAGEMENT

The code was designed with features to prevent hazards. One safety feature is operator control. The operator can turn **fuzzymeter** on and off, switch between metering algorithms, and tune parameters. Another safety feature is fall-back modes to continue operation in the event of a failure. For instance, if the fuzzy metering rate cannot be calculated for some reason (such as insufficient data), **fuzzymeter** is automatically disabled at that location. Operation falls back to **bottleneck** if it is enabled, or local metering if **bottleneck** is not enabled. If no central metering rate is provided either from **fuzzymeter** or **bottleneck**, the local logic within the 170 is used. Table 5 lists possible risks posed by the new software and steps to mitigate those risks.

Table 5: Risk Mitigation

Risk	Steps to Mitigate Risk
Software bugs that crash system or produce errors	Extensive software quality tests will be performed during the installation phase to ensure that code works properly and is bug-free. The log files will be examined for possible errors.
Improper timing of Vax-170 communications	Vax-170 communications will be tested first with a simulation field rack and then a single ramp to verify proper protocol. The maximum time needed for data communications will be estimated and timed to verify a sufficient margin of safety.
Improper timing of TAPS execution	The maximum time needed for the executions of all TAPS will be estimated and timed to ensure a sufficient margin of safety.
Non-ideal metering rates executed	The minimum and maximum for each parameter, including metering rates, provide a safety check to make sure the rates are within a reasonable range. Algorithm parameters will be tuned to optimize performance measures. Performance will be compared to the bottleneck and local metering algorithms through the FLOW evaluation.
Poor response to unusual conditions such as data errors, incidents, special events, construction, and extreme weather	The algorithm is designed for robustness under a broad range of conditions. With the use of descriptive and comprehensive inputs, the controller handles both recurrent and non-recurrent congestion. The measures of congestion that are fed into the controller take into account reduced capacity, whether it is caused by incidents, weather, or construction. Sensitivity to inaccuracies in the data are mitigated by the use of multiple detectors, fuzzy preprocessing of the data, and parallel rule evaluation. Operators have the ability to switch between central ramp metering algorithms (fuzzymeter versus bottleneck), turn metering on and off, and switch between *Time Of Day* and *Central* ramp metering.
Excessive queues	In the event of an excessive queue, the operators can easily tune the algorithm from the operator console window. To reduce the queue, operators can increase the weighting factors for the queue override and advance queue override inputs.
Lack of operator knowledge about new software	TSMC personnel should be trained to operate, tune, and maintain software. A user manual and documentation of software will be provided. TSMC personnel will be given the opportunity to ask questions and make suggestions during implementation.

CONCLUSIONS AND RECOMMENDATIONS

This research project made progress toward implementing a fuzzy ramp metering algorithm for the Northwest Region of the WSDOT. On-line implementation turned out to be a non-trivial task for several reasons. The consequences of any modification were studied comprehensively to avoid problems, particularly because the TSMC performs vital functions with wide-scale implications. The new software was carefully designed so that it is easy to use, consistent with the existing TSMC software, and highly reliable. Additional tasks to those in the original proposal were necessary for successful implementation, such as software documentation, design of the operator interface, and refinements to the testing plan.

Study of the TSMC software took much longer than anticipated because of the complexity of the software, insufficient documentation, and lack of WSDOT knowledge of the code. To help future software modifications proceed more smoothly for other researchers, we make three recommendations:

1) Require software documentation. For future modifications, software documentation should be required. For portions of the pre-existing code that are studied, documentation should be provided to assist others.
2) Designate a resident expert at the WSDOT to make or assist with code modifications.
3) Assess the cost of contracting out major software modifications to Jerry Hautamaki at HNTB, who wrote the original code for the TSMC VAX. Although his hourly cost is steep, it may be worthwhile given how long it takes to become familiar with the code.

To understand and design the **fuzzymeter** interface, it was necessary to document the portions of the TSMC software that interface with the fuzzy controller. The **build_rmdb** software is where the fuzzy parameters and fuzzy equations are entered. The

RMDB structure needed alteration to include the new fuzzy parameters. The real-time processes were studied to interface **fuzzymeter** in a parallel manner to the other TAPS. The error handling and logging functions were studied to make sure that **fuzzymeter** could use them. The operation of **rmdc_comm** was examined to determine the best way to send the metering rates to the 170s.

The main product of this research project is the new code that was written for the fuzzy ramp metering algorithm and its interface with the TSMC software. The code design emphasizes reliability through data usability, levels of defaults, and fall-back modes to continue operation in the event of a failure. The new code includes 10 new modules for the algorithm. Sixteen pre-existing modules also required modification to interface the algorithm. The modified modules relate to **build_rmdb**, the RMDB, and the **rmdc_comm**. Modification of the VAX-170 communications was necessary to properly implement metering rates directly.

The operator interface design emphasizes algorithm tunability and control features. The new code allows the operator to enter and tune new parameters, specify fuzzy equations, switch between ramp metering algorithms, and observe the controller inputs and outputs. It also logs errors and events to the appropriate output file. The specifications for the new database elements were designed. Instructions for tuning parameters and entering fuzzy equations were given.

For successful implementation, it was necessary to develop a plan that would thoroughly evaluate software integrity and algorithm quality. The testing plan will verify that the new software is performing properly and that the on-line testing is done uniformly. Performance measures and testing sites were decided upon.

The training technique, architecture, and data sets of the neural network predictors were improved. Despite these improvements, the predictions lack accuracy when generalizing to new data sets. Because the predictors are difficult to train for each site and because prediction accuracy is inconsistent, the neural networks are not recommended for

implementation at the present time. If time and resources allow, the neural network predictors will added later as inputs to the fuzzy ramp metering algorithm. If not, the fuzzy ramp metering algorithm has comprehensive inputs that serve the same function as the predictive inputs and that performed well in simulation.

Despite the fact that the implementation is not progressing as quickly as planned, the most difficult obstacles have been surmounted, such as understanding the pre-existing TSMC software and designing the interface between the fuzzy ramp metering algorithm and the TSMC code. The researchers will proceed with implementing and testing the new algorithm beginning September 1997.

REFERENCES

L. Chen and A. May, "Freeway Ramp Control Using Fuzzy Set Theory for Inexact Reasoning," *Transportation Research-A*, Vol. 24A, No. 1, 1990, pp. 15-25.

D. Meldrum and C. Taylor, "Freeway Traffic Data Prediction Using Artificial Neural Networks and the Development of a Fuzzy Logic Ramp Metering Algorithm," Final Technical Report, Washington State Department of Transportation, Olympia, WA, 1995.

J. Robinson and M. Doctor, "Ramp Metering Status in North America: Final Report," Office of Traffic Operations, Federal Highway Administration, U. S. Department of Transportation, Washington, D.C., Sept. 1989.

H. Taale, J. Slager, and J. Rosloot, "The Assessment of Ramp Metering Based on Fuzzy Logic," Proceedings of the Third Annual World Congress on Intelligent Transport Systems, Orlando, Florida, 1996.

C. Taylor, "Freeway Traffic Data Prediction Using Artificial Neural Networks and the Development of a Fuzzy Logic Ramp Metering Algorithm," Master's Thesis, Department of Electrical Engineering, University of Washington, 1994.

C. Taylor and D. Meldrum, "Simulation Testing of a Fuzzy Neural Ramp Metering Algorithm," Final Technical Report. Washington State Department of Transportation, Olympia, WA, October 1995.

APPENDIX A. 1. FUZZYMETER

fuzzymeter [./rt_skeleton/fuzzymeter.c] starts up main, builds fuzzymeter analysis table, and then waits for event flag from real time skeleton to calculate metering rates. Fuzzymeter is integrated as a Traffic Analysis Program (TAP), and is designed in a parallel manner to the other TAPS.

- **general_process_startup**
  - **connect_to_mailbox**
  - **write_to_crash_log**
  - Associate to event flag cluster
  - Clear all event flags
- **log_tms_event**
  - **log_tms_common**
    write_to_crash_log
    get time
    compose message
    write_to_mailbox_nowait
- **map_to_RTDB**
  - **map_to_global_section**
  - **init_rtdb_tl**
- **map_to_RMDB**
  - **map_to_global_section**
  - **init_rmdb_tl**
- **build_fuzzymeter_table (see below)**

- <u>While</u> SHUTDOWN_TMS event flag is clear
  - Wait for fuzzymeter start event flag with SYS$WAITFR
    This flag gets set in rt_skeleton by SYS$SETEF
  - Clear fuzzymeter start flag with SYS$CLREF
  - **calc_fuzzymeter (see below)**
  - Set fuzzymeter done flag with SYS$SETEF
  - Test shutdown flag with SYS$READEF
    cond_code of SS$_WASCLR means don't shutdown
    cond_code of SS$_WASSET means shutdown

build_fuzzymeter_table - Parses the fuzzymeter equation file (which was created by build_rmdb and must adhere to the format specified in Section IV.F.), searches for cabinet name in RMDB and station names in RTDB, and writes action codes and data pointers to fuzzymeter table. (see following fuzzymeter table description)

- Initial memory allocation for fuzzymeter table
- write header to table: start table label, number of sets, table size, date/time
- open fuzzymeter equation file
- Initialize counters and flags

- While not EOF, read line of FUZZY_METER.EQN (equation file)
  Example of 1 set (where a set is an equation):
  ES-158R:MMS_FM1 = ES-158R:MMS_Stn | ES-156R:MMS_Stn & ES-154R:MMS_Stn | ES-160R:MMS_Stn & ES-161R:MMS_Stn | ES-158R:MMSQ_1 & ES-158R:MMSI_1 | ES-158R:MMSRA_1(2) & & ES-158R:MMSLA_1(4)

  - Get cabinet name at start of equation
  - Initialize location to LOCAL
  - Save pointer to beginning of set
  - Initialize error flag to no error
  - **find_fddb_cl_name** -- Search for cabinet name in Field Device Data Base (FDDB) and return index
  - Extract metered lane number from the cabinet:loop name
  - If cabinet name is not in RMDB, write error message and set error flag
  - If "FM" is not given after cabinet name to fuzzy meter, write error message and set error flag
  - If lane number is not a digit, write error message and set error flag
  - If an error occurred, send error message and search for beginning of next equation
  - If more memory is needed to write next set, allocate additional memory
  - Write set header to table: start set code, lane number, space for # of bytes in set, pointer to cabinet in RMDB (for fuzzy parameters)
  - Get token from eqn_file

  - While token is a delimiter (not end of equation)
    - If token is '|', increment location
    - Get next token. Expecting a station:loop name
    - If a token not returned
      build_tap_error -- station:loop name not found
      Break out of while-loop to skip equation (Error handling at end of while loop resets pointer to the beginning of set and looks for next equation)
    - If location is ADV_QUEUE
      - Parse token with strtok to get the station:loop name
      - If station:loop name is not found, set error flag and break out of while-loop to skip this equation

- Parse same token with strtok to get number of samples
- If station:loop name is not found, set error flag and break out of while-loop to skip this equation
- Convert # of samples from ascii to integer
- If # of samples is less then 128, convert it to an unsigned char (1 byte)
- Else
  - build_tap_error -- # of samples is too large
  - set error flag
  - Break out of while-loop to skip this equation
- Write location code to table
- **search_rtdb_name_table** -- search for index to loop in RTDB
- If loop index is not found
  - build_tap_error -- Stn/Loop name is not in RTDB
  - set error flag
  - Break out of while-loop to skip this equation
- Obtain offset in table for stn_loop in RTDB based on index
- Write offset to table
- If location is ADV_QUEUE
- Write # of samples to table
- Get next token -- expecting a delimiter or cabinet
- If next token is NULL (OK if EOF)
  - build_tap_error -- Null result when parsing fuzzy equation
  - set error flag
  - Break out of while-loop to skip this equation
- End of while-loop that reads equation

- If location is not ADV_QUEUE
  Point to set start to skip this equation because it does not have the right number of station locations. (fscanf has already grabbed the next token.)
- Else if no errors occurred (the equation is good)
  - Increment the number of sets in the table
  - Calculate the number of bytes in the set
  - Write the # of bytes in set to table
- Else an error occurred during parsing (NULL fscanf result)
  - Point to set start to skip this equation
  - Get token to search for next equation
- End of while-loop to read file of equations

- Close fuzzy equation file
- Write number of sets and table size into table header
- Write table end label, check sum (calc. # bytes in table)
- Trim table size
- Return base address of table

FUZZY_TABLE -- Description of what it looks like in memory

LABELS	ITEM	# BYTES	DESCRIPTION
			(begin table)
table_base->	code	1	FM_TABLE_START
	ushort	2	number of sets
	ulong	4	table size
	date_time	8	struct system_time
			(end of table header)
			(beginning of sets)
set_start	code	1	FM_SET_START
	byte	1	lane_no -- which ramp lane to meter (1, 2, or 3)
	byte	1	# of bytes in set
	ulong	4	col_ptr to cabinet in RMDB (to get fuzzy parameters)
			(end of a set header)
			(begin SR stations)
	byte	1	LOCAL -- adjacent mainline station type
	ushort	2	rtdb_offset to LOCAL station in RTDB
			(end of this station)
	byte	1	DOWN -- downstream station type
	ushort	2	rtdb_offset to downstream station in RTDB
			(repeat for each downstream station)
	byte	1	UP -- upstream station type
	ushort	2	rtdb_offset to upstream station in RTDB
			(repeat for each upstream station)
	byte	1	QUEUE -- ramp queue station type
	ushort	2	rtdb_offset to queue loop in RTDB
			(repeat for intermediate queue if given)
	byte	1	ADV_QUEUE -- station type
	ushort	2	rtdb_offset to advance queue loop in RTDB
	byte	1	# of samples to calculate adv_queue_occ
			(repeat for each advance queue)
			(end of a set)
			(begin next set)
Repeat for each set (every cabinet with fuzzy metering)			
			(end of all sets)
			(label end of table)
	code	1	TABLE_END
	short	2	check sum -- # of bytes in table
			(end of table)

calc_fuzzymeter -- When called from main every 20 seconds, process the fuzzymeter table 1 line at a time, obtaining fuzzy parameters from RMDB and getting data from RTDB. After parsing a set (for a metered ramp), Call calc_fuzzy_rate, which returns metering rate. Write it to RMDB. The new data poll sends direct metering rates and the 170 bypasses local logic to directly implement them.

- For all data columns in RMDB
  - Skip if data column for min, max, default or prot mask
  - Initialize the Fuzzy Metering Rate to zero for lanes 1, 2, and 3. The 170 interprets 0 to mean that the fuzzy metering is disabled for that lane. Bottleneck isn't initialized to disabled because we may want to use bottleneck and fuzzy meter on different lanes within the same cabinet. If bottleneck is enabled for the cabinet, it will be used on the metered lanes for which the fuzzy metering rate is disabled.
- Initialize pointer to beginning of fuzzy meter table
- Skip past table header

- While not end of table
  - Save pointer to beginning of current set in table
  - Get metered lane # from table
  - Get bytes in set from table
  - Get station:loop offset into RTDB
  - Get pointer to data column in RMDB
  - Use column pointer to get index to current cabinet
  - Get Permit Fuzzy Metering parameter from RMDB
  - If Permit Fuzzy Metering parameter is disabled,
    - Skip this set and jump pointer to the next set in table
    - Get action code that starts new set
    - Continue to beginning of while-loop
  - Get fuzzy parameter high and low range limits for each input
  - Get fuzzy parameter rule weights
  - Get range limits for Queue Occupancy, Advanced Queue Occupancy, and Metering Rate. These names are lane specific, so only get the parameters for the current lane.
  - Initialize the centroid and base width fuzzification parameters to their default values, which are provided with the globals at the beginning of fuzzymeter.c
  - Get next action code

  - While processing RTDB station:loops
    - Initialize data usability flag to yes
    - Initialize number of good station:loops for this data type to zero
    - Initialize sum of volumes for this station type to zero
    - Initialize sum of scan count for this station type to zero
    - If action code is QUEUE
      number of samples used for queue occupancy input is 5
    - Else
      number of samples used for other inputs is 1

- Do-While same station type (action codes are the same)-- loop executed at least once. For each time this loop is executed, one input to the fuzzy controller is calculated (see rules for writing fuzzy equations in documentation for details)
  - Get station:loop offset into RTDB from table
  - If action code is ADV_QUEUE
    Get number of samples used to calculate advanced queue occupancy input
  - For each sample needed to calculate input data
    - Calculate pointer to RTDB cabinet from station:loop offset
    - **unpack_rtdb_loop_stn** data
    - If the number of loops is greater than 1 (this means that it's a station, not a loop) and the flag not equal to 0 (the station data is usable)
      OR If the number of loops is 1 (loop data rather than station) and the flag is 1 (the loop data is good)
      Note: This data has already been interpolated if it is necessary and possible
      - Increment number of good stations used to calc. this input
      - Count number of total loops (over all stations) to calc. this input
      - Sum scan count
      - If action (detector location) is LOCAL or DOWN
        Sum volume to later calculate speed
  - Get next action code
- End of do-while loop to process stations of same type

- If more than one station:loop is good, calculate controller inputs
  - Calculate average occupancy using scan count
  - If location code is LOCAL or DOWNSTREAM
    Calculate average speed for that location using average volume and occupancy
  - Switch based on old action code (detector location)
    Enter the calculated occupancy and speed (if used at that location) in controller input array
- Else (data for this location is insufficient to calculate inputs) -- Check to see if the lack of data at this location makes the rule base incomplete. If the rule base is incomplete, set the data_usable flag to NO so that Local Metering will be used instead.
  - Switch based on old action code (detector location)
    - case LOCAL:
      Data is not usable -- a complete rule base requires this input
    - case DOWN:
      Set the rule weights to zero for the rules that use this input (the rule base does not require this input for completeness). These rules include DownSpVs-OccVb, DownSpS-OccB, and DownSpVs-OccVb
    - case UP:
      Set the rule weights to zero for the rules that use this input (the rule base does not require this input for completeness). These rules include UpOccM, UpOccS, and UpOccVs.
    - case QUEUE:

Set the rule weight to zero for the rule that uses this input, QueueOcc. If there are no ramp inputs (both QueueOcc and AdvQueueOcc data inputs are unusable), the rule base is incomplete. Because the resulting fuzzy metering rate may be too restrictive, Use Local Metering instead.

- case ADV_QUEUE:

  Set the rule weight to zero for the rule that uses this input, AdvQueueOcc. If there are no ramp inputs (both QueueOcc and AdvQueueOcc data inputs are unusable), the rule base is incomplete. Because the resulting fuzzy metering rate may be too restrictive, Use Local Metering instead.

- End of while-loop to process a set (an equation for 1 metered lane)

- If data is usable to calculate the metering rate at this ramp
  - **calc_fuzzy_rate** given the real-time data, fuzzy parameters, and rule weights
  - If the metering rate is greater than 25.5, it won't fit into 1 byte

    Write error message with **log_tms_event**
  - Convert the metering rate from a float to an unsigned character byte
  - Write the metering rate for that lane to the RMDB
- Else if data is not usable to calculate the metering rate at this ramp
  - Write error message with log_tms_event
- End of while-loop to process table

calc_fuzzy_rate -- This function returns a metering rate given inputs to the fuzzy controller. If the fuzzy ramp metering algorithm were ported to other systems, this function and those that it calls would be used because they contain the fuzzy ramp metering algorithm. All other functions are interface for the fuzzy controller, and unique to this TSMC.

- **fuzzify** the inputs give the fuzzy parameters -- this translates each input into a set of five fuzzy classes
- Evaluate **rules** given fuzzy inputs and rule weights
- **defuzzify** rule outputs into a single numerical metering rate

Fuzzify -- Converts each numerical input into a set of five fuzzy classes. For each input, it calculates, the array of fuzzy classes that indicate on a scale of 0 to 1 the degree to which each class is true.

- For each input
  - If high dynamic limit is lower than the low dynamic limit
    write error message with **log_tms_event**
  - Calculate Very Small fuzzy class -- The class is 1 if the rescaled input is less than 0, and the class is 0 if the rescaled input is greater than the base width for the Very Small class. In between, it's a linear relationship.
  - For Small, Medium and Big classes
    Calculate fuzzy input -- It's a triangular class (see tunable parameter definitions in documentation for details)
  - Calculate Very Big fuzzy class -- The class is 1 if the rescaled input is greater than 1, and the class is 0 if the rescaled input is greater than 1 minus the base width for the Very Big class. In between, it's a linear relationship.

Rules -- Evaluate each rule and return a set of five fuzzy classes for the metering rate.

- Evaluate each rule given the fuzzy input. The degree of the rule outcome is equal to the minimum of degrees in the premise. This corresponds to a logical AND between two rule inputs.
- Multiply each rule outcome by its rule weight
- Calculate weighted sum of rule outcomes for each class

Defuzzify -- Use discrete fuzzy centroid to convert set of fuzzy metering rates to a single numerical metering rate

- For each fuzzy class of metering rate
  - Calculate the implicated area of the fuzzy rule outcome. (See documentation on tuning parameters for details. This is the fuzzification process in reverse).
  - Calculate the centroid of the implicated area of fuzzy class
  - Accumulate numerator sum for discrete fuzzy centroid-- the area of fuzzy class times the centroid of fuzzy class times the sum of weighted rule outcomes for that class
  - Accumulate denominator sum for discrete fuzzy centroid -- the area of fuzzy class times the sum of weighted rule outcomes for that class

- If denominator is too small
  Write error message with **log_tms_event**
- If high_limit is not greater than low_limit for metering rate dynamic range limits.
  Note: the resulting metering will be between these limits.
  Write error message with **log_tms_event**
- Calculate metering rate = num/den and rescale from the (0,1) range to the (low_limit, high_limit) range
- If metering rate is now within the (low_limit, high_limit) range
  Write an error message with **log_tms_event**
- Return metering rate

APPENDIX A. 2. CHANGES TO BUILD_RMDB

Build_rmdb opens and reads rmdb_input.fil, builds RMDB, creates temporary files (loop_names.lst, inc_det.eqn, btl_neck.eqn, speed_traps.lst, stn_aggr.eqn, station_names.lst, actv_anal.eqn) which are later used to build tables in global memory for traffic analysis programs. It also sorts names, loops, stations and speed traps and writes them to "rtfmdbname.srt" to be used for later creation of RTDB and FMDB. Build_rmdb appears deceptively simple, but in fact, it starts a long chain of events, calling function upon function. For details on how build_rmdb works, see TSMC software documentation.

To incorporate the fuzzy ramp metering algorithm, build_rmdb must also read the fuzzy parameters and fuzzy equations from rmdb_input.fil and create a temporary file called fuzzy_meter.eqn, which is subsequently used by fuzzymeter to build the fuzzy table in global memory.

The only changes made to build_rmdb.c itself are to open the temporary equation file "fuzzy_meter.eqn" before reading rmdb_input.fil and to close this file after reading rmdb_input.fil. However, there are several changes in functions which are indirectly called from build_rmdb.c.

Modify **process_input_special_case** in **fddb_sub.c** [located in /fddb/fddb_sub.c]

Called from **get_param**. A pointer to the function **get_param** is in the read_fddb_file, which contains a function state table that tells **read_fddb** (called from build_rmdb) how to parse the input file rmdb_input.fil.
process_input_special_case calls functions to handle the current input line depending on the current group index. The only change to **process_input_special_case** is to call **get_fuzzy_eqn** when the current group index is FUZZY_EQNS. These are the two new functions to handle the two new groups (see next page).

RMDB_SUB.C [located in /fddb/rmdb/rmdb_sub.c]-- Two new functions added: **get_fuzzy_eqn** and **get_next_fuz_line.**

Note: Fuzzy parameters are not a special parameter case, so they are handled by **load_param**, which is called from **get_param.**

get_fuzzy_eqn [/fddb/rmdb/rmdb_sub.c] -- Parses current line from data file and writes it to a fuzzy equation file in a fixed format.

Note: The return values are used differently than they are for the other functions called from **process_input_special_case.** END_LINE is returned regardless of error or success so that get_param does not continue to parse the same line. If error, the message is logged by fddb_error and the line is skipped by returning END_LINE.

Note: See documentation on how to write fuzzy equations in Section IV.F.

- Calculate pointer to data column
- Make sure data column is a RAMP_MTR or DATA_STN, otherwise write error message and return to get next line
- Get cabinet:loop name from line buffer using strtok.
  Note: strtok uses 1 or more skip characters as delimiters between tokens. strtok returns the pointer to the next token in the input buffer and writes a NULL at the end of the token. Subsequent calls using NULL as the first argument continue parsing the same buffer and remember the current location
- Verify that cabinet:loop name is in proper format with **get_cab_loop_name**
- If the cabinet name does not match the current group name
  Write error to **fddb_err** and return to get next line
- Initialize current pointer to beginning of a buffer to be written to fuzzy equation file.
  Note: Before writing the reformatted equation to the fuzzy equation file, it is written to a temporary buffer. This allows writing over the equation (skipping it) if an error is found.
- Write cabinet:loop name to buffer and update pointer
- Initialize number of loops written to this line to 1

- For each station location
  - Initialize the number of station/loops at this location to zero

  - Do-while stations are of same type (the loop executes the first time, and continues to execute as long as the '&' delimiter is between station names)
    Note: '&' is used to delimiter between two stations of the same location type, and 'l' is used to delimiter stations of different location type.
    - If the allowable number of loops is exceeded, write error message to fddb_error
      Up to two stations are allowed for the QUEUE input (see documentation notes)
      Only one station is allowed for the LOCAL input
      The remaining locations (UPSTREAM, DOWNSTREAM, and ADV_QUEUE) allow up to five detectors.
  - Get cab:loop name from input line using strtok

- If token is not found
  - Get loop detector (or station) name from input line using strtok
  - If no token found
    - get next line from input file with **get_next_fuz_line**
    - If line_type is not parameter, write error message to **fddb_error** and return to get next line
    - Try again to get loop detector (or station) name from input line using strtok
  - Verify that cabinet:loop name is in proper format with **get_cab_loop_name**
  - If the cabinet name does not match the current group name
    Write error to fddb_err and return to get next line
  - Increment the number of loops found for this station
  - If the number of loops already written to this line on buffer is 4, begin on next line because this line is full
  - Write detector name to buffer and update buffer pointer
  - Increment the number of loops written to this line in buffer
  - Get delimiter from input line using strtok (expecting '&' or 'l' to continue equation)
  - If token is not found
    Write error message with fddb_error and return to get next line
  - Else if token is '&'
    Write delimiter to buffer and update pointer
- End of do-while loop

- If delimiter is not equal to 'l', there is a missing delimiter. (The equation was expected to continue)
  Write error message with **fddb_error** and return to get next line
- Write delimiter to buffer and update pointer
- End for each station location

- Do-while processing ADV_QUEUE loop names
  - Get cab:loop name from input line using strtok
  - If token is not found
    - Get loop detector (or station) name from input line using strtok
    - If no token found
      get next line from input file with **get_next_fuz_line** (see next page)
    - If line_type is not parameter, write error message to fddb_error and return to get next line
    - Try again to get loop detector (or station) name from input line using strtok
  - Verify that cabinet:loop name is in proper format with **get_cab_loop_name**
  - If token not found
    Write error to fddb_err and return to get next line
  - Get number of samples from input line using strtok
    This indicates the number of previous samples used to calculate this input
  - If the number of loops already written to this line on buffer is 4, begin on next line because this line is full
  - Write detector name to buffer and update buffer pointer
  - Get delimiter using strtok

- If a delimiter was found and it was equal to '&' (the equation continues)
  Write delimiter to buffer and update pointer
- End of do-while processing ADV_QUEUE loop names -- No more stations means that it is the end of the equation.

- If delimiter is not NULL, this means there was an extra token(s) at the end of the equation that is not proceeded by a delimiter.
  Write error message with fddb_error and skip over extra tokens
- Write the buffer to fuzzy equation file
- Return to get next line of input file

get_next_fuz_line [/fddb/rmdb/rmdb_sub.c] -- This new function is called from **get_fuzzy_eqn** when a fuzzy equation continues past more than one line.

This function is identical to **get_next_btl_line** [located in /fddb/rmdb/rmdb_sub.c] except for the error message. Although the same function could have been used for both with slight modification, changes to bottleneck were avoided.

- Loops until line type returned by **get_next line** is a blank, form feed, or comment
  get_next_line returns line_type of comment, curly_brace, square_bracket, or parameter. (See documentation of build_rmdb for details). Line is stored in global memory tl->lb_ptr->line_buffer
- If finds wrong line_type, writes error
- Else returns with parameter type line

APPENDIX A.3. Changes to RMDB Structure

To incorporate the fuzzy ramp metering algorithm, additional global parameters must be stored in the Ramp Metering Database (RMDB). Changes to the RMDB are made in rmdb.h (structure changes) and rmdb_tbl.c (array values).

Changes to rmdb.h [in /fddb/rmdb/rmdb.h]

- Define indices for two new group names
  - #define FUZZYMETER_PARAMS 20
  - #define FUZZYMETER_EQNS 21
- Modify the struct rmdb_file_pointers (this is one of the tables pointed to from the structure rmdb_table_list) to include a FILE pointer for fuz_eqn_file. This is for the temporary equation file created by build_rmdb and used by fuzzymeter to build the fuzzy table.
- Modify the data column structure to include the new fuzzy parameters, which include dynamic range limits, rule weights, operator permit fuzzy control, and the resulting fuzzy ramp metering rates for each lane. The parameter specifications are given in Section IV.C.
- Define indices to the new fuzzy parameters

Changes to rmdb_tbl.c [in /fddb/rmdb/rmdb_tbl.c]

- Modify the minimum, maximum, default, and ep_mask arrays to include initial values for the new parameters. These arrays are of structure type rm_dc_data_col defined in rmdb.h. The arrays must fit this structure.
- Add new parameters to the name_table. This array is of type fddb_name_table, which is defined in fddb.h. The structure itself does not need modification. This array must correspond with parameter structure for data column in rmdb.h.
- Add new group names and element names to output_list array. This array is of the structure fddb_output_list, defined in fddb.h. The structure itself does not need modification. The output list must correspond with parameter structure and group indices defined in rmdb.h.
- Add the two new group names to the group_table array. This array is of the structure fddb_group_table, defined in fddb.h. The structure itself does not need modification. This array must correspond to the group indices defined in rmdb.h.

APPENDIX A. 4. Changes to RMDC_COMM

Without software modification, there is not a way to directly implement a metering rate to the 170. The existing bottleneck algorithm produces a metering rate adjustment, which is sent through the data poll. The 170 makes further adjustments based on local conditions.

We have explored two options for directly implementing the metering rate produced by fuzzymeter:

1) Set the minimum and maximum allowable metering rate to be equal to the desired metering rate
2) Modify the VAX-170 communications protocol to send the metering rate directly.

We recommend that the second method be used for several reasons. It turns out that implementing the metering rate by the first method is nontrivial. While method 1 might be acceptable for test purposes, method 2 is desirable for final implementation, so we would need to do it eventually. It is recommended that we proceed with method 2 because it is more straightforward. A discussion of the changes necessary for each method follows.

Method 1

- Because the minimum and maximum metering rates would be written over by the desired metering rate, we need a way to store the minimum and maximum rates elsewhere so we do not lose this information. The easiest solution to this problem is to locally store the minimum and maximum metering rates for all data columns in fuzzymeter. Each time **calc_fuzzy** is called, the "real" minimum and maximum metering rates are written back into the RMDB for all ramp meters. Then if the fuzzy rate is enabled for a given ramp, the calculated metering rate writes over the minimum and maximum values.
- To update the minimum and maximum metering rates within the 170, Load Parameters (which contain these rates) must be sent to the 170 every 20 seconds. Load Parameters 1 contains an 8 byte header, followed by 84 bytes of parameters. The response from the 170 contains 8 bytes. This technique is not ideal due to the additional communications required, although it is feasible within the system constraints.
- Load Parameters 1 are not automatically sent every time a change is made in one of these parameters. Here is a summary of the additional functions required to transmit and flag this change:
  - **update_rmdb** [in fuzzymeter.c] is called from **calc_meter_rate** (see fuzzymeter docu-

mentation of this function)

- Writes the metering rate to RMDB.
- Checks to see if the port number is valid,
- Checks to see if the unit is not in communications failure
- Checks to see if the unit is enabled,
- Checks to see if the port is enabled.
- If these checks are passed, the update flag for LDP1 is set, indicating to RMDC_COMM that load parameters 1 need to be sent to the 170.

- **add_update_to_list** [in fuzzymeter.c] is called to build a list of commands that must be sent to RMDC_COMM to update each 170.
- **mail_cmds_to_rmdc_comm** [in fuzzymeter.c] is called to transmit the command list to RMDC_COMM. This function uses mailbox messages to communicate between mains.

Method 2

This method requires changes to the communications protocol between the VAX and 170.

This involves minor code modification of rmdc_comm_sub.c, rmdc_comm.c, and the 170.

Changes to **rmdc_comm_sub.c** [/fddb/rmdb/rmdb_sub.c]

- **build_and_queue_170_meter_rate** -- This new function creates a new data poll which can directly send metering rates and places the data poll on buffer to be transmitted at end of queue. This data poll would replace the existing data poll. The first 8 bytes look identical to the current data poll, with the bottleneck adjustment sent in byte 5. Direct metering rates for lanes 1, 2, and 3 are sent in additional bytes 8, 9, 10 (1 byte for each lane). Bytes 11 and 12 are the data CRC bytes.

Switching between central ramp metering algorithms:

- With this new data poll, bottleneck and fuzzymeter can operate simultaneously on different lanes within the same cabinet. For this reason, it would be redundant to use a switch to indicate whether bottleneck or fuzzymeter is in effect for that cabinet. Instead, a fuzzy metering rate of zero indicates that it is disabled for that lane. If the fuzzy metering rate is disabled, the bottleneck metering rate is used on those lanes, unless it is also disabled. If both central algorithms are disabled, the local metering rate is used.
- Fuzzymeter is similar to bottleneck in that both algorithms disable the central metering rate before writing over it. Every time bottleneck is called, it writes -128 to the metering rates for all cabinets to disable them. Then it writes over the ones for which OperPermitBtl is enabled. Likewise, fuzzymeter writes zero into MeterRateLane1, 2, and 3. If OperPermitFuz1 is enabled, it overwrites a nonzero metering rate into lane 1.
- Essentially, if OperPermitFuz1, 2, or 3 is enabled, it overrides the bottleneck adjustment. In general, inexplicit override of one controller over another is undesirable, but it is unavoidable in this case because fuzzymeter is lane specific while bottleneck is cabinet

specific.

- In the future, it may be desirable to add a third central metering algorithm and have the capability to choose between them. It would not be difficult to add this capability. Additional parameters will need to be added to data columns within RMDB, including flags to indicate which algorithm to use in each lane and storage space for the metering rates generated by each central algorithm. If logic were to be used to choose a metering rate, this new main would be treated as an additional TAP. Because each TAP is completed before the next begins, logic to choose metering rates could use the results from fuzzymeter, bottleneck, and any additional algorithms.

- In **build_and_queue_170_msg**, change name of command DATA_POLL to DATA_POLL_RATE_ADJ (line 2908). Call the new command name DATA_POLL_METER_RATE
- Changes to the **manual_menu** function
  - Call **build_and_queue_170_meter_rate** instead of **build_and_queue_170_msg** when polling is selected from the menu (line 6566)
  - Add a declaration for a RMDB column pointer and a calculation of this pointer

Changes to **rmdc_comm.c** [/fddb/rmdb/rmdb_comm.c]

- Add case for when command byte is DATA_POLL_METER_RATE. This case is handled identically as for DATA_POLL_RATE_ADJ (line 2908)
- For both multi mode and test mode, call **build_and_queue_170_meter_rate** instead of **build_and_queue_170_msg**

Changes to 170 logic

- Method 2 requires minor changes to the 170 logic to correctly interpret the new data poll. Because fuzzymeter directly implements a rate, the new logic must bypass the local logic when this occurs. This design allows the existing logic to remain in place and operate as before, unless the fuzzy metering rate is provided. The pseudocode for the new logic looks like this:

```
For each lane
    If fuzzy metering rate > 0 (it is not disabled)
        Use fuzzy meter rate
    Else
        Proceed with pre-existing logic for bottleneck/local algorithms
```

- Note that any central ramp metering algorithm which needs to directly implement a rate can use this same mechanism by sending it through the new data poll. It is not limited to use by fuzzymeter.
- The 170 must know to interpret the fuzzy metering rate as an unsigned byte ranging between 0 and 25.5 VPM (the bottleneck adjustment ranges between -12.5 and 12.5).

To save time, it is recommend that the 170 change be done either in-house at WSDOT, contracted out, or parcelled out to a student that we hire.

APPENDIX B. OPERATOR INTERFACE

The operator interfaces to fuzzymeter in the same way as the other TAPS (like bottleneck): through the rmdb_input.fil, opc_comm, patch_rmdb, watch_rmdc, watch_fuzzymeter, and log files.

1. RMDB_INPUT.FIL is used to build the RMDB off-line.
   - This is where fuzzy parameters which differ from the compiled default value are specified. The hierarchical default specification is the same for fuzzy parameters as for other RMDB parameters. See Section IV.D. for fuzzy parameter descriptions and default values. See section IV.E. for parameter tuning instructions. See TSMC software documentation for a detailed explanation of the default hierarchy.
   - This input file is where fuzzy equations are written. There must be a fuzzy equation for each metered lane that uses the fuzzy ramp metering algorithm. The equation specifies the cabinet and lane number to be metered, and which stations or loops to use for each input to the controller. The rules for writing equations are given in Section IV.F.

2. OPC_COMM allows the operator to control the fuzzy ramp metering algorithm. Any changes to OPC_COMM to display and modify fuzzymeter variables will be done in a parallel fashion to bottleneck.
   - Fuzzy parameters can be displayed and modified using the Meter Tuning Window.
   - Inputs to the fuzzy controller can be viewed through the Single Meter Status Box.
   - Start and stop fuzzy metering through the Meter Control Box by choosing ON or OFF.
   - The Lane Status Box identifies the type of ramp metering in effect. Fuzzy parameters will need to be added to this list.
   - Switching between fuzzy metering and bottleneck metering is done by modifying the parameters OperPermitFuz1, OperPermitFuz2, and OperPermitFuz3 (one for each metered lane) from the Meter Tuning Window. Because bottleneck and fuzzymeter can operate simultaneously on different metered lanes within the same cabinet, there is not an explicit switch to indicate which central algorithm is in use for that cabinet. When Operator Permit Fuzzy is disabled, bottleneck is in effect for that lane, unless it is also disabled. If both fuzzymeter and bottleneck are disabled or if the ControlSwitch is set to TOD, the local logic takes over. For a complete explanation of switching between algorithms, see Section IV.A.4.

To save time, it is recommend that PC software changes be done either in-house at WSDOT, contracted out, or parcelled out to a student that we hire.

3. PATCH_RMDB

Patch_rmdb is useful for making parameter changes from the VAX without rebuilding the data base. Fuzzy parameters will be added to the menu of group names so that any fuzzy parameter can be changed from patch_rmdb. The fuzzy equations can only be changed through rmdb_input.fil, which requires rebuilding the RMDB. Only 1 version of patch_rmdb can run at a time.

4. WATCH_RMDC

Watch_rmdc dumps real time data to the screen for the device requested. This feature is useful to observe inputs to the ramp metering algorithm. RTDB values can also be viewed through the operator console.

5. WATCH_FUZZYMETER

A new main called watch_fuzzymeter will be used to observe the inputs from loop detectors and the resulting metering rates of the fuzzy logic controller for a given ramp. This code will be modelled similar to watch_bottleneck. Watch_bottleneck duplicates the code of bottleneck to reproduce the same calculations. Watch_fuzzymeter will attempt to avoid this redundancy by accessing the fuzzymeter calculations.

6. Output Files

- Operator Log
  Some of the operator log information relates to ramp metering, including when meters are turned on, turned off, enabled, or disabled. If loops are reset, enabled, or disabled, this is written to the operator log. This log can also be used to see when incidents occurred.

- Journal File
  The journal file is written to every time a change is made to the RMDB from the operator console. (If the changes are made through PATCH_RMDB, they are not written to the journal file). The journal file was designed to be read upon building the RMDB and incorporate the latest changes. However, it does not work properly. Nevertheless, it does contain a record of RMDB changes made, and thus, may be useful for debugging purposes in the event of a system crash.

- TMS Event Logger
  This log records events and errors that occur related to TAPS (but it is not limited to TAPS). If a bug occurs related to fuzzymeter, this log should be checked for error messages.

- Communications Handler Log File
  This records events and error messages related to the communications handlers. If problems occur with the VAX-170 communications, they should be investigated here.