PERFORMANCE MEASURE CALCULATION USING HIGH-RESOLUTION DATA

A Thesis

Presented in Partial Fulfillment of the Requirement for the

Degree of Master of Science

with a

Major in Civil Engineering

in the

College of Graduate Studies

University of Idaho

by

Randal Brunello

March 2014

Major Professor: Michael P. Dixon, Ph.D., P.E.

AUTHORIZATION TO SUBMIT THESIS

This thesis of Randal Brunello, submitted for the degree of Master of Science with a Major in Civil Engineering and titled "Performance Measure Calculation Using High-Resolution Data" has been reviewed in final form. Permission, as indicated by the signatures and dates below, is now granted to submit a final copy to the College of Graduate Studies for approval.

Major Professor:	_____ Date: _____	
	Michael P. Dixon, Ph.D.	
Committee Members:	_____ Date: _____	
	Michael Kyte, Ph.D.	
	_____ Date: _____	
	Axel Krings, Ph.D.	
Department Administrator:	_____ Date: _____	
	Richard Nielsen, Ph.D.	
Discipline's College Dean:	_____ Date: _____	
	Larry A. Stauffer, Ph.D.	

Final Approval and Acceptance

Dean of the College of Graduate Studies:	_____ Date: _____	
	Jie Chen, Ph.D.	

ABSTRACT

High-Resolution data collected by traffic signal controllers can be used to create performance measures. This project produced a tool that could take VISSIM output files and create a high-resolution data table in a database. Then the tool can calculate performance measures and output required values into an excel file. These values are calculated and reported on a cyclic basis so that they may be plotted and displayed together. The calculated measures were tested using manual observation and data collection points. The only measure that was not 100% accurate at calculating the performance measure was the delay and queue length estimation. This measure requires very precise data and is a promising candidate for continued research.

ACKNOWLEDGEMENTS

This project would not have been possible without the support and guidance of the faculty and staff of NIATT. To everyone that helped me I would like to say thank you very much. I couldn't have done it without you. A special thanks to:

Dr. Michael P. Dixon, my major professor. Thank you for guiding me through my research and supporting me through the conclusion of this project. Your help with the writing of the thesis has been invaluable.

Dr. Michael Kyte, my committee member, for sparking my interest in transportation engineering and being a leading contributor to my graduate education.

Dr. Axel Krings, my committee member, for taking time out of your busy schedule to be a last minute committee member. I hope you don't cringe too much at the programming.

My parents for helping support me through both my undergraduate and graduate studies. I couldn't have made it this far without all of your love and care.

Victor House for helping with the programming and database work at multiple stages along the way.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

BOG – Beginning of Green

EOG – End of Green

GUI – Graphical User Interface

GTU – Green Time Utilization

PCD – Purdue Coordination Diagram

PPTC – Purdue Phase Termination Chart

VISSIM - Verkehr In Städten - SIMulationsmodell (German for "Traffic in cities - simulation mode")

VBA – Visual Basic for Applications

UDOT – Utah Department of Transportation

Chapter 1 INTRODUCTION

1.1 Overview

The purpose of this research project was to create a tool that could be used to facilitate future research of traffic signal systems. The tool has four functions that calculate performance measures using high-resolution data collected by traffic signal controllers. Each function produces an excel file with data laid out such that it can easily form a graph. The measures are normalized and calculated for each cycle so they can be displayed versus time. This normalization of measures allows multiple measures to be included on a single graph. With multiple measures shown this way they can be analyzed more effectively. The first function produces an Excel file that can generate a Purdue Coordination Diagram (PCD) as well as the percent arrivals on green, percent green time, and platoon ration. The second function produces an Excel file that can display the Green Time Utilization (GTU) of a phase. The third function produces an Excel file that can generate a Purdue Phase Termination Chart. The final function calculates an estimation of delay and queue length and produces an Excel file that can display these measures. Along with the current performance measurement, it can be expanded to use multi-source data such as Bluetooth probe data or speed detection. The other primary function of this tool is to facilitate the ability to easily use micro simulations to generate high-resolution data. This will help with the creation of new performance measures by removing the necessity of hardware-in-the-loop simulation.

Chapter 2 LITERATURE REVIEW

2.1 Introduction

The purpose of this literature review is to describe the common performance measures that can be generated using high-resolution or similar forms of data. The performance measures below are all included in the generated tool. These measures are continuing to be improved and are recommended for future research. In conjunction with one another these measures can identify most operational problems at traffic signals as well as help find a solution. As such, they are prime candidates in a prototype performance monitoring tool.

2.2 Purdue Coordination Diagram

Darcy Bullock at Purdue University is the leading source on high-resolution performance measures. The Purdue Coordination Diagram (PCD) is one of his creations (Brennan, 2011). This Diagram is a chart that displays arrivals in relation to the time in the cycle, and the cycle in relation to the time of the day. Only one phase is analyzed on a single diagram. See Figure 2-1 below for an example. It is a chart that displays arrivals in relation to the time in the cycle and the cycle in relation to the time of the day. It is used to analyze the quality of coordination, among other system monitoring activities. Each point on the diagram is a vehicle arriving at the intersection. If it is on the lower division of the graph it is an arrival at the intersection during red. The green line indicates the beginning of green (BOG). If it is on the upper division of the graph it is an arrival during green. The red line at the top of this division indicates the end of green (EOG). A well-coordinated system will show vehicle arrivals as a dense cloud, as indicated by the “ii” note in Figure 2-1.

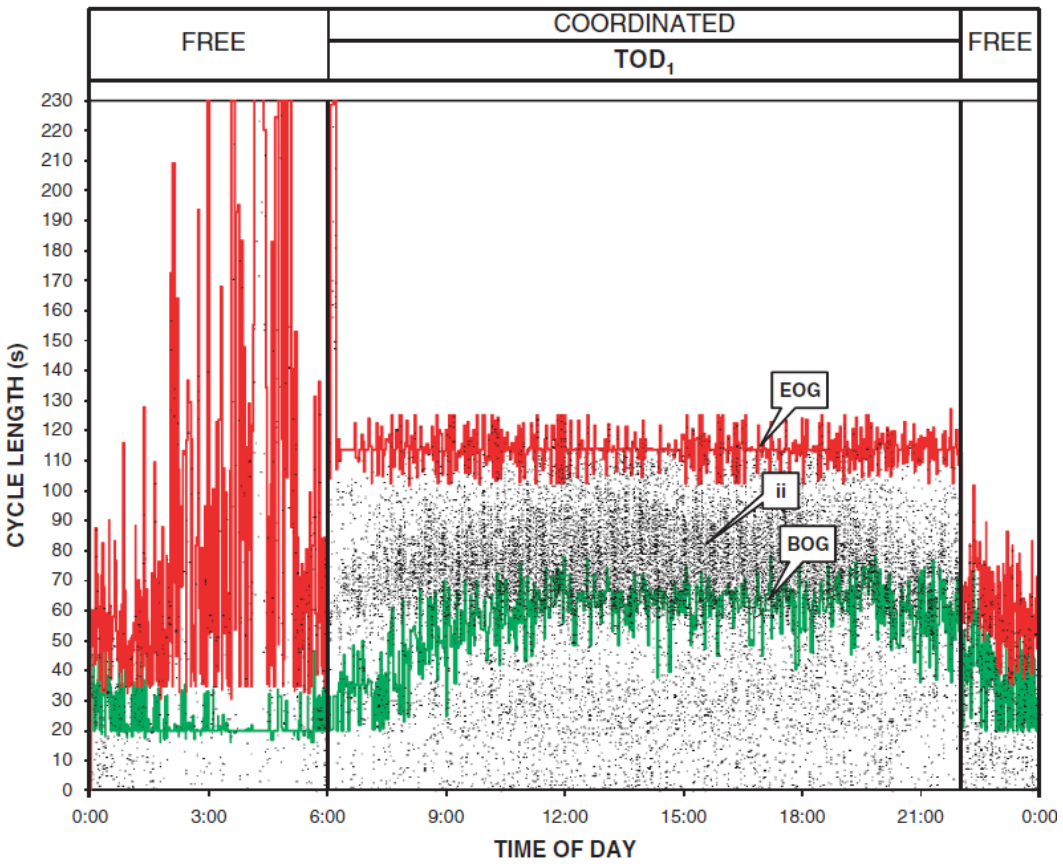


Figure 2-1. Purdue Coordination Diagram (Brennan, 2011)

The PCD does take a trained eye to be analyzed. Issues like queues extending over the advance detectors and the lack of numerical quantification can make using the PCD challenging. For this reason Utah Department of Transportation (UDOT) includes average performance measures for each time of day plan. The percent arrival on green, percent green time, and platoon ratio give a numerical quantification that is a good representation of one of their diagrams. The percent arrival on green is calculated as the number of arrivals during the green time divided by the total arrivals during that study period. Similarly the percent green time is simply the total green time divided by the total time of the study period. The platoon ratio combines these two measures and is calculated as the percent arrivals on green divided by the percent green time.

The PCD sheds light on a broad range of traffic operations problems, like queue spillback or poorly timed offsets. At the same time, the PCD integrates well with more aggregate measures. As a result, the proposed tool should include the PCD to help advance performance measurement by generating multiple performance measures, while leveraging the PCD to see individual vehicle operations.

2.3 Purdue Phase Termination Chart

The UDOT has implemented an extensive system that logs high resolution data for the majority of the traffic signals on the state system (UDOT, 2013). Their system allows users to view performance measures depending on the detection setup. For example if the intersection has advance detectors, PCDs can be generated. Most high-resolution performance measures depend on detectors. The Purdue Phase Termination Chart is the only measure that does not require detector data. This chart, see Figure 2-2, shows the condition that lead to the green ending. It color codes max-outs, gap-outs, and force-offs. Then comparing different phases it can help identify problems with split times and in some cases find malfunctioning detectors. For instance, Figure 2-2below shows a plot created on UDOT's performance metrics website. The green dots are gap-outs, the blue dots are force-offs and the red dots are max-outs. Since the max-outs are similarly colored to the pedestrian walk markers, a group of max-outs has been circled in the figure. The orange dots that are just above the phase are pedestrian walk indications. There are two coordinated periods, one in the AM and the other in the PM. Because of the consistent force-off phase termination, it is clear that phases 2 and 6 are the major approach that is coordinated. During plans 1 and 13 where there is coordination they are always forced off.

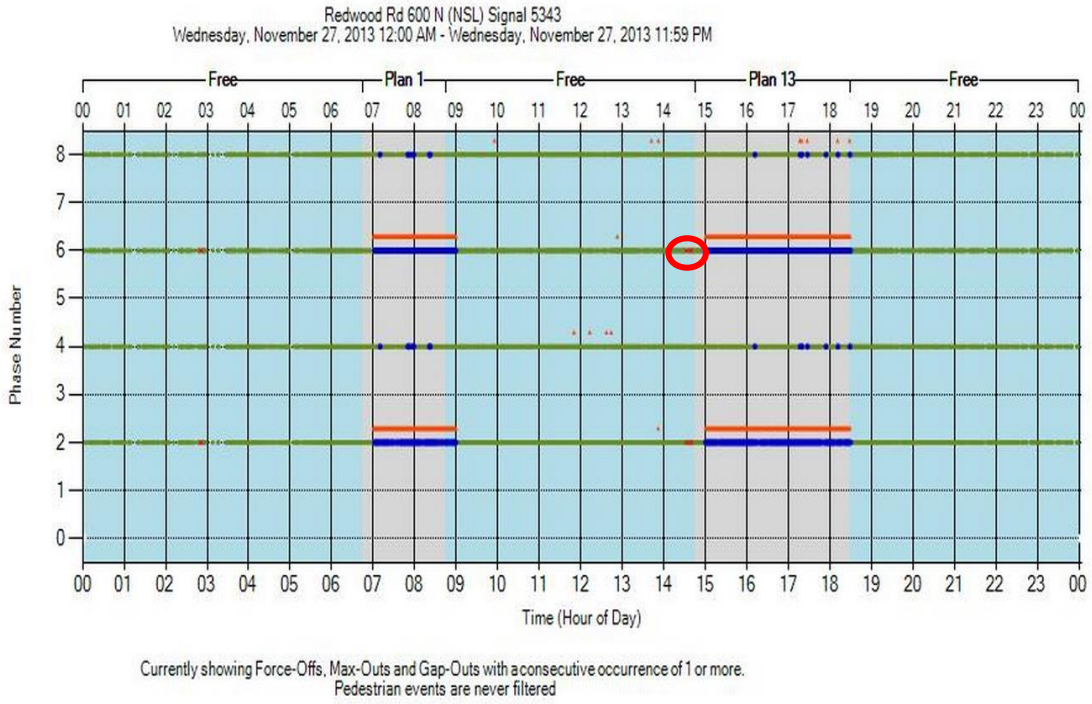


Figure 2-2: Purdue Phase Termination Chart (UDOT, 2013)

This measure is very simplistic but can still be very useful. For instance, in the case shown above, if another measure identified a problem on the major corridor this diagram could be consulted to compare the different phases. Due to the consistency of the minor phases gapping out, this intersection would be able to have its split times adjusted to better meet its demands. Unfortunately nothing concrete can be determined from this measure alone, it is simply an assistant to the other measures.

2.4 Green Time Utilization

Green Time Utilization (GTU) is a simple performance measurement of the percent green time that the stop bar detectors were active. This serves as a surrogate measure for capacity utilization. However, it requires calibration for proper use. Calibration is necessary because the variety of detection types, accuracy, and sensitivity affect the GTU significantly. However, even after being

calibrated it still does not have a significant correlation with delay. A study found that GTU had an R^2 of .513 when correlated to delay, (Smaglik, 2011). It can be a good indicator of the performance even though it is not very precise. In their study, when the GTU was greater than 95%, half of the cycles were split failures, meaning the queues failed to clear during the green time (see Figure 2-3). Note that Smaglik used the term green occupancy ratio (GOR) instead of GTU.

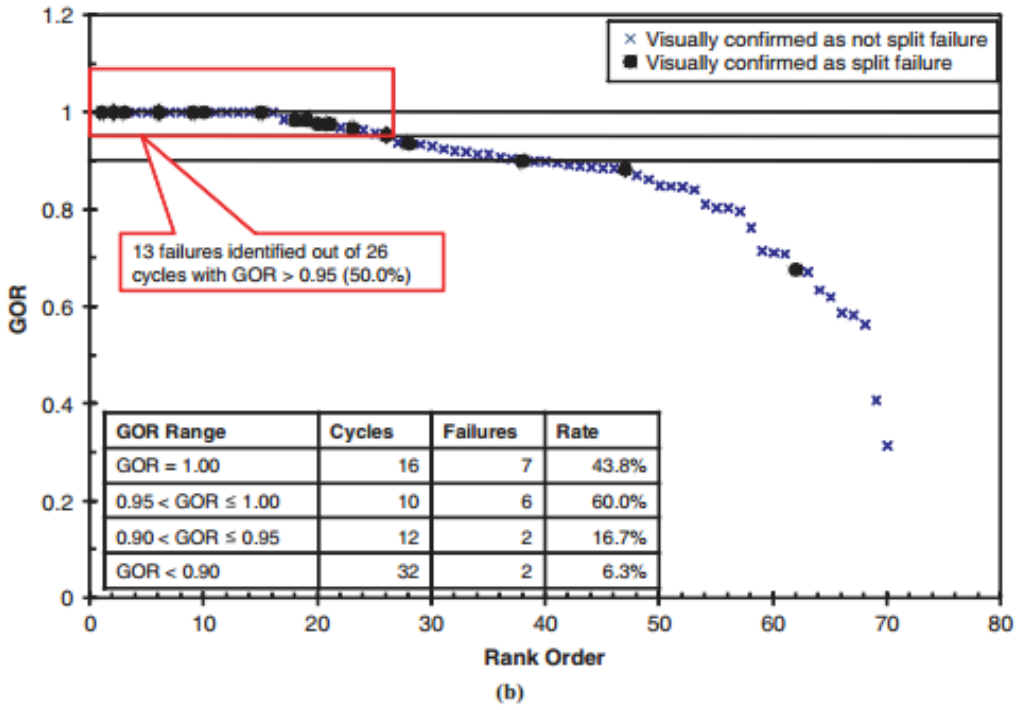


Figure 2-3: Green Occupancy Ratio and Split Failures (Smaglik, 2011)

2.5 Delay Measurement

UDOT also did a study on estimating delay. It used loop detector data to estimate delay and travel times using stop bar data from the target intersection and the upstream intersection. The estimated travel time would determine which detector hits from the downstream intersection would refer to which detector hits in the upstream intersection. The dataset was trimmed to not include vehicles that were already in the system at the start of the analysis or still in the system at

the end of the analysis. They also had to account for vehicles exiting and entering mid segment. This problem led to the creation of two methods. The first method had detectors on the mid segment driveways so that the vehicles that exited and entered could be accounted for. The other method did not have these mid segment detectors. The second method is the only one that might be viable for use in the tool. Once the vehicles entering and exiting mid-stream have been removed from the analysis, the process checks to make sure that all the data points from the upstream detectors have matches with data points from the downstream detectors. Then the average travel time is calculated. First the sum of the upstream detector hit times is subtracted from the sum of the downstream detector hit times. That value is then divided by the total number of vehicles. The average delay time is then defined as the difference between the average travel time and the ideal travel time, the time it takes a vehicle to drive through the test zone while driving the speed limit. The results of this study seemed promising with errors in delay less than 5 seconds per vehicle for most cases. Unfortunately, that is value is out of context. The percent error varied from a significant 30 to 40 percent. This is caused by the distance between measurements. The longer the distance the more complex the model for the drivers' behavior needs to be. Intersection-to-intersection requires more than the simple expectation that drivers will travel the speed limit. The delay will still be used in the tool, but the distance between entrance and exit detectors will be much shorter, approximately 400 feet.

Chapter 3 METHODOLOGY

3.1 Introduction

The system for generating performance measures requires 2 parts. The first part is a database that contains the high resolution data and, for conversion of VISSIM, traffic micro simulation software developed by PTV group, outputs to high resolution data, a table that contains some of the signal timing parameters. The database used in this project was Microsoft SQL Server. The other part is the collection of Python programs which perform the necessary calculations and produce the output. There are a total of seven files that make up the performance measure suite. Only two of the files need to be run or edited to use the program. Only, the file that produces the graphical user interface (GUI) and the file that controls the default information are needed. In order to run the different performance measure programs all that must be done is the opening of the GUI program. In order to change the default information the Python file named “CustomDefaults.py” must be edited.

3.2 High-Resolution Data Emulation

This program converts VISSIM output files to basic eventcode style high resolution data. High resolution data is a record of every change of state. It includes the time stamp of the state change, the event code to identify the state that is changing, the signal identifier, and the phase identifier. The Detector output file (.ldp) and the signal output file (.fzp). The detector output files must be converted to (.csv) files to be read by the program. Any form of control, including hardware in the loop, is acceptable, so long as the same output files are created.

3.2.1 Interface GUI

The interface for this tool is shown below in Figure 3-1. The input fields can all be customized using the CustomDefaults.py. The GUI starts by requesting a study number. The study number has an effect on the tables that are created in the database and despite its name it does not need to be a number. So long as the study number is unique it will not have an effect on the operation of this program. The next set of information requested by the emulator is the signal numbers. These are entered with a comma separating each signal. The signal numbers must be integers and must match the signal numbers in the simulation. They are not constrained by order outside of the previous. Once the signals have been entered the detectors for each signal must be entered. The current program is prepared for one to four signals. While the fields are labeled Intersection 1-4 they just are linked to the numbers entered into the Intersections field, which do not need to be 1-4. An increase above four signals must be hardcoded into the program for it to be able to run. The detector numbers must be entered into the field matching the order of the detector data in the VISSIM output files. If there are fewer than four intersections the values in the extra intersections

detector fields will be ignored by the program. The final entry is the name of the table that the data needs to be output to. This must be a unique name.

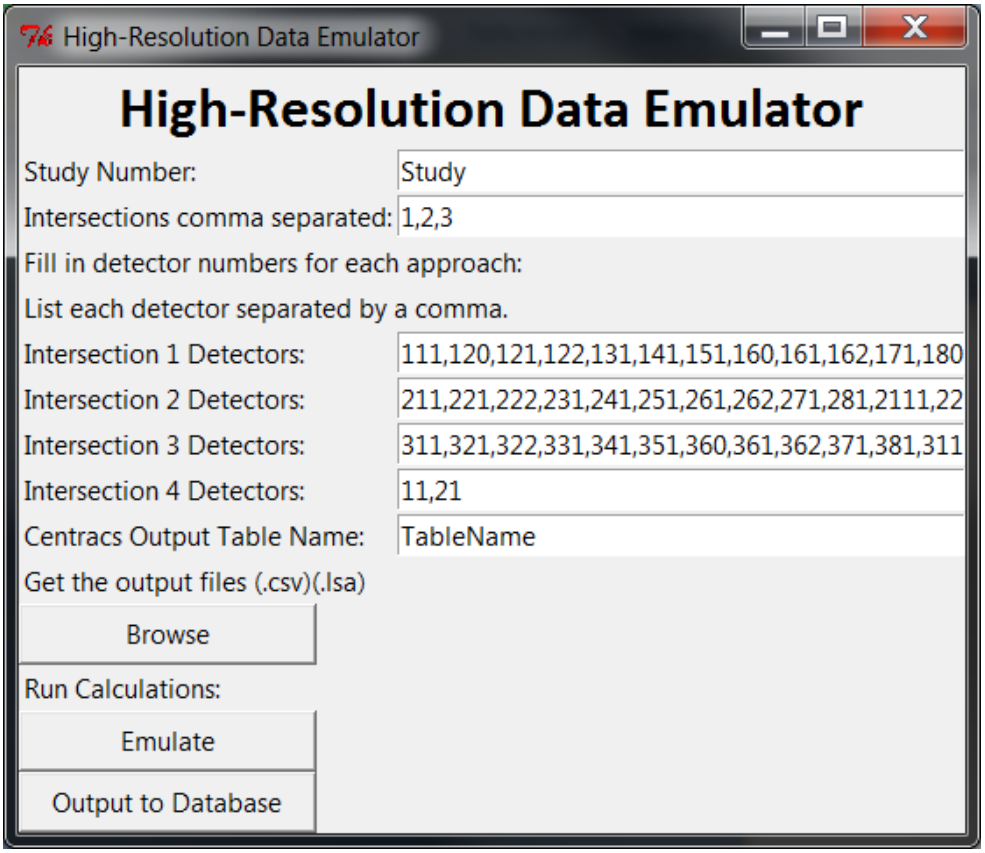


Figure 3-1. Simulation to High-Resolution Data Emulator

There are three buttons that follow the entry fields. The first button is a browse button that will collect the simulation output data. First the detector data is collected in the order of the signals entered into the input field. Following the detector data's collection the signal changes are collected. These files are then each entered into a table in the database. The next button runs the analysis and conversion of the output files to high resolution data. The detector data is output by VISSIM every time step. Therefore, it must be simplified down to detector activation and deactivation times. The signal changes are already output by VISSIM in the form of state changes.

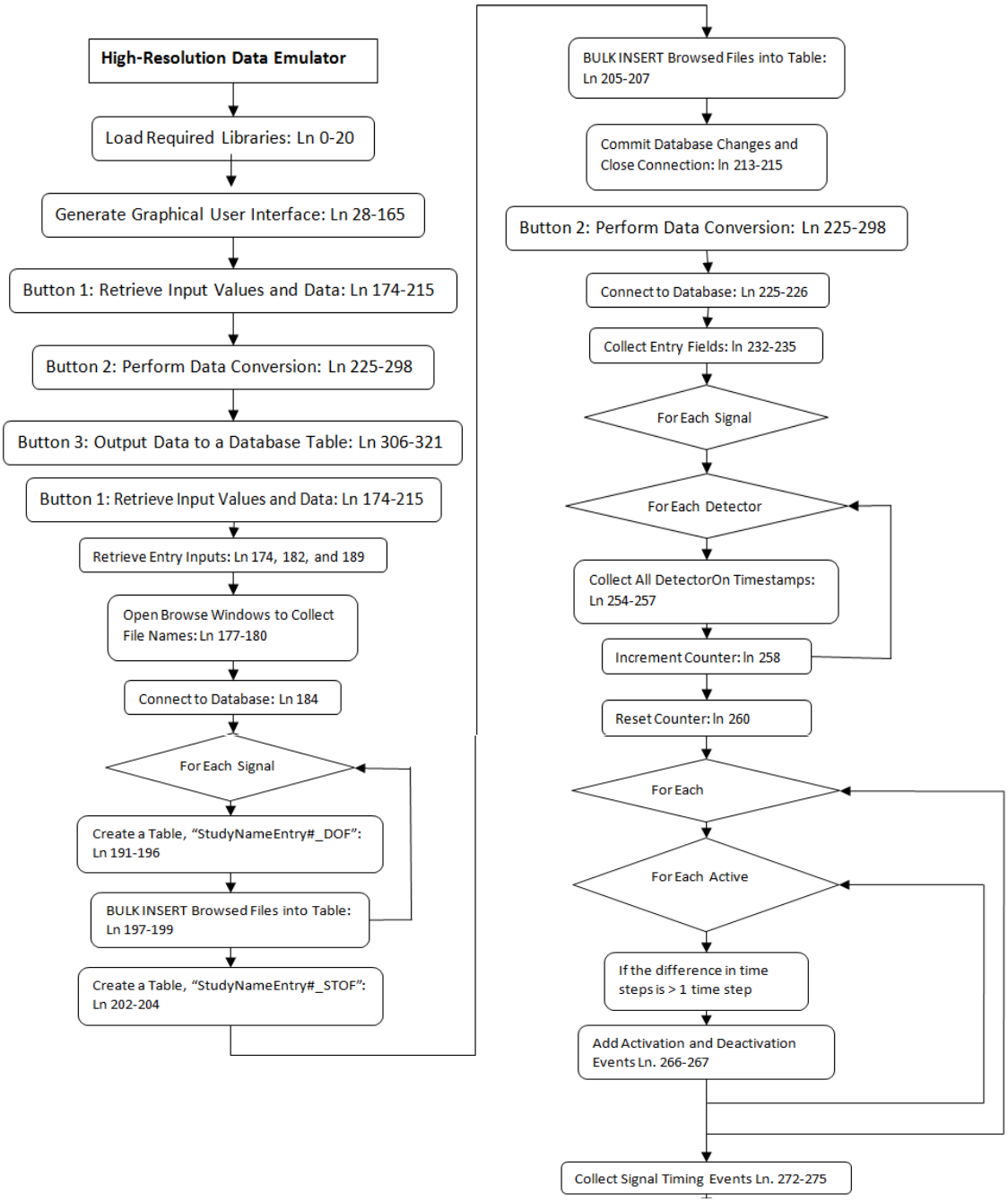
Only the format must be changed to fit the event code style. Also, the green times are checked with the maximum green to determine the phase termination. The maximum green times are retrieved from a table in the database. The table is formatted to have 3 fields, the intersection, phase, and maximum green. These fields were integers in the example but could be any numerical data type that is needed. Each file is analyzed and entered into a string. This string is then entered into the database. Since the order of the data produced by this program is not chronological, all queries to this table need to include an "ORDER BY TimeStmp".

3.2.2 Flow Chart Narration

Figure 3-2 explains each step through the program. Each of the functions has its own smaller flow chart since they do not need to be run in order, although it is recommended.

This program has four sections of code. The first section of code generates the GUI. This consists of seven entry fields and three buttons. The first entry is used to name tables in the database that store the output files for this simulation. Each time the first function is run, by pressing the Browse button, this field will be used. It must have a unique value if a new dataset is being entered into the database. However, the first function can be skipped if the dataset has already been run through it. Just leave this first entry as what it was when the dataset was added and skip to the second button. The second entry collects the signal numbers. Any number may be entered and they do not need to follow any particular order. The values must be separated by commas. The following four entry fields collect the signal detector information. The four fields are hardcoded into the program. A system with fewer signals will simply ignore the extra detector entry fields. A system with more will require some slight adjustment to the program. Any numbers may be entered but they will be associated with the data from the detector files in the order they are entered. Therefore, it is recommended that these values be taken directly from the detector output files. The last entry

field is the name that the database will be saved as. This value needs to be unique. If there is a table with this name already the program will have an error. If the program fails to run it is most likely either a problem with Python libraries or these entries.


High-Resolution Data Emulator
Load Required Libraries: Ln 0-20
Generate Graphical User Interface: Ln 28-165
Button 1: Retrieve Input Values and Data: Ln 174-215
Button 2: Perform Data Conversion: Ln 225-298
Button 3: Output Data to a Database Table: Ln 306-321
Button 1: Retrieve Input Values and Data: Ln 174-215
Retrieve Entry Inputs: Ln 174, 182, and 189
Open Browse Windows to Collect File Names: Ln 177-180
Connect to Database: Ln 184
For Each Signal
Create a Table, "StudyNameEntry#_DOF": Ln 191-196
BULK INSERT Browsed Files into Table: Ln 197-199
Create a Table, "StudyNameEntry#_STOF": Ln 202-204
BULK INSERT Browsed Files into Table: Ln 205-207
Commit Database Changes and Close Connection: ln 213-215
Button 2: Perform Data Conversion: Ln 225-298
Connect to Database: Ln 225-226
Collect Entry Fields: ln 232-235
For Each Signal
For Each Detector
Collect All DetectorOn Timestamps: Ln 254-257
Increment Counter: ln 258
Reset Counter: ln 260
For Each
For Each Active
If the difference in time steps is > 1 time step
Add Activation and Deactivation Events Ln. 266-267
Collect Signal Timing Events Ln. 272-275

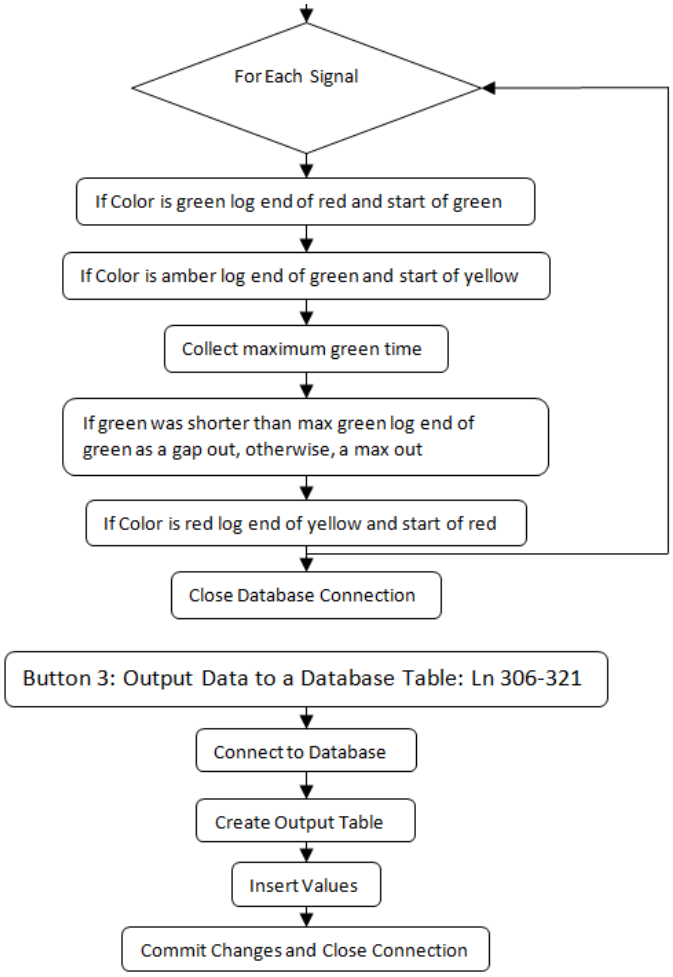


Figure 3-2. Simulation to High-Resolution Data Emulator

The second section of code retrieves the VISSIM output files and enters their data into the database. It starts by opening a browse window. It will ask for a detector output file for each signal that was entered into the signals entry field as well as the signal timing output file. Once the output files have been located the program reads the files into separate database tables for easier access.

The third section of code uses the previous tables that were created and converts their format to state-change events. This section of the program has the most complex process. It starts by retrieving values from the input fields, in case the second section did not need to be run. Then the program loops through the signals and prepares the detector events. The VISSIM output provides the status of the detector each time step. This must be changed to detector activations and

deactivations. To do this all of the active times are retrieved and processed. When the difference between the previous active timestamp and the current active time stamp is more than one time step then the previous active timestamp plus one time step is logged as a deactivation and the current timestamp is logged as activation. Once the detector output information has been analyzed the signal timing information is analyzed. This information is already in state change and only requires one logic check. When the signal state changes to amber the previous green time is checked to determine if it was a max-out/force-off or a gap-out. All values are placed into a string which is used in the next section of the program.

The final section of code creates a table using the input from the final entry field as a name. It then deposits the entirety of the string produced in the previous step into the table, commits the changes, and closes the connection.

3.2.3 Coding Guidance for Anticipated Improvements

3.2.3.1 Expanding Number of Intersections

Currently the program is hard coded to only allow four intersections because there are only four intersection entry fields. There are two avenues for improving this. The first option is to open a second window that will retrieve the detector information. This can be generated with a loop and is the only way to automate this process. The second option is less complicated but more stringent. The number of fields can simply be increased by copy-pasting sections of code, making small changes to that code, and larger changes to other code down the line. If there is a maximum number of intersections that are likely to be used it is recommended that the second alternative be used. If there is uncertainty in the size of networks the first alternative is significantly more flexible but will require more up-front time.

The first alternative will use code similar to a section from the turning count estimation. The function Hits in this program will need to be copied into the program. The hard coded fields will all need to be removed and a button for the function Hits will be added in their place. The locations where the detector fields are retrieved will have to be changed to loops to match the new flexibility. This should be done using append on the lists where this information is stored. The CustomDefaults.py file will have to be adjusted to allow for this flexibility. The simplest option will be to have a single default for all fields that requires the operator to adjust. Otherwise, there will have to be a hard-coded number of defaults which will limit the flexibility of the program.

The second alternative will require that a detector label and entry be copied and pasted until the number of signals goal has been reached. The row count variable will take care of the formatting but the entry and label objects will need to be renamed. Also, the locations where the detector fields are retrieved will need to be adjusted to match the new number of signals. If the values attribute of the entries is updated then the CustomDefaults.py file will need to be expanded to match. This does not need to be changed but can greatly increase the ease of using this program on a single system.

3.2.3.2Output from Other Micro-Simulation Software

Changes in output file format will be difficult to address without significant code changes. If there are only minor changes, such as detector status changed to ‘true’/’false’ or ‘1’/’0’ instead of ‘|’/’.’, then the change will be simple. If the output files are in the form of status changes already it would likely be easiest recode all the logic, only maintaining the writing of events to the string. The “if” statements in the loop that analyzes the signal timing output file will require adjustment. The string they check for is very specific, requiring the exact number of advance and trailing spaces. It may even be easier to write a separate program that changes the output files to match what is required.

3.2.3.3Incorporation of External Data

Including other forms of data, like Bluetooth probe detection or manually collected data, is not difficult. As long as you can produce an event code entry from external data it is a simple matter to add it to the database. If it is going to be occurring in every study it may be included in the current functions. More likely however, this process would be added as a separate function that runs upon button press. This would require the external data to be read into the program, either by reading a text file, not the process used in any other part of this program, or by inputting a file into the database and reading it from there. Then looking through the data and appending events to the same string that is produced by the emulation. This can be done entirely with processes already developed within the program.

3.3 Purdue Coordination Diagram

3.3.1 Introduction

This program analyzes high resolution data and outputs an Excel file that contains data organized into columns. This Excel file can easily be used to create a Purdue Coordination Diagram, or PCD, using a scatter plot. It also produces summary measures including: percent time green, percent arrivals on green, and platoon ratio for each cycle and overall. These cycle-by-cycle measures are plotted along with or on the PCD.

3.3.2 Interface

This interface, seen below in Figure 3-3, is the base interface for most of the following programs. It has four fields that collect information. As with all of the programs, the default values for the graphical user interfaces (GUI) can be changed using the CustomDefaults.py file. The first field is the signal numbers separated by commas. The second field is the phase numbers separated by commas. Only one phase number can be selected for each intersection selected. However, the

same intersection may be listed multiple times to include multiple phases from one intersection and any number of intersection/phase pairs are acceptable. Together the first two fields must produce pairs of intersection and phase numbers, as marked below. The detector input field is slightly more complicated. The detectors per phase are separated by commas. Then each phase's group of detectors is separated from the other phases' groups of detectors by a semicolon. The final entry is the database table where the data will be retrieved. At the very bottom of the GUI there is a button which will run the program once the fields have been prepared. This button will run the entire program from retrieving inputs to creating the Excel output.

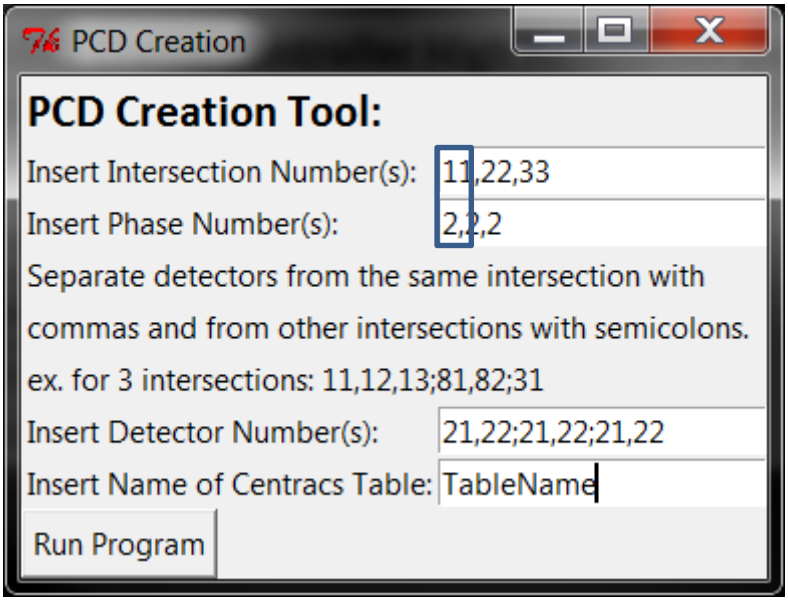


Figure 3-3. Purdue Coordination Diagram Interface

3.3.3 Flow Chart Narration

The Purdue Coordination Diagram program has two parts and these are illustrated in Figure 3-4. In the first part, it constructs the graphical user interface, seen in Section 2 of the flow chart. The second part runs the analysis based on the information entered into the graphical user interface, seen in Section 3 of the flow chart.

The user interface is described in detail above, so this discussion emphasizes the calculation process, which begins by retrieving the information from the graphical user interface. Then the process connects to a database and creates an Excel workbook, Steps 3.3 and 3.4 in Figure 3-4. Once this has been done, it loops through the intersection/phase combinations that were entered into the graphical user interface. For each loop, the program uses the remaining input information to construct three SQL queries, Step 3.6. The queries retrieve all of the intersection green times, red times, and detector activations for the given phase. These are stored in three separate lists. The process organizes the data cycle-by-cycle by looping through the beginning-of-red time list. The beginning-of-red time indicates the ending time of the current cycle for which detection data are being organized. For each cycle's red time, the program loops through the detector times, searching for detections that occurred during the cycle. The loop through the detection times stops when a detector time occurs after the beginning-of-red time. Each detector time is logged as an arrival in the workbook. Next, after completing looping through the detector data the program checks the green time list to see if the green time is the correct one. It does this check by making sure that the green time occurred between the previous beginning-of-red time and the current beginning-of-red time. It then logs the green time in the workbook for the current cycle for which data are being organized. Once it is done looking at the detector times and green times, it calculates the cycle's performance measures. Percent arrivals on green, percent green time, and the platoon ratio are the measures calculated. The program then moves to the next cycle's red time and repeats this process until there are no more red times. Once all the red times have been completed, the program steps to the next intersection and repeats the process again. Last the program saves the workbook and it is ready to be used to create a PCD. All output excel files are saved in the same directory as the python programs. The excel files are named as the performance

measure and datetime separated by an underscore. For example, the format, PerformanceMeasureName_Month_Day_Hour_Year, results in a PCD_Jan_15_2_30.xlsx.

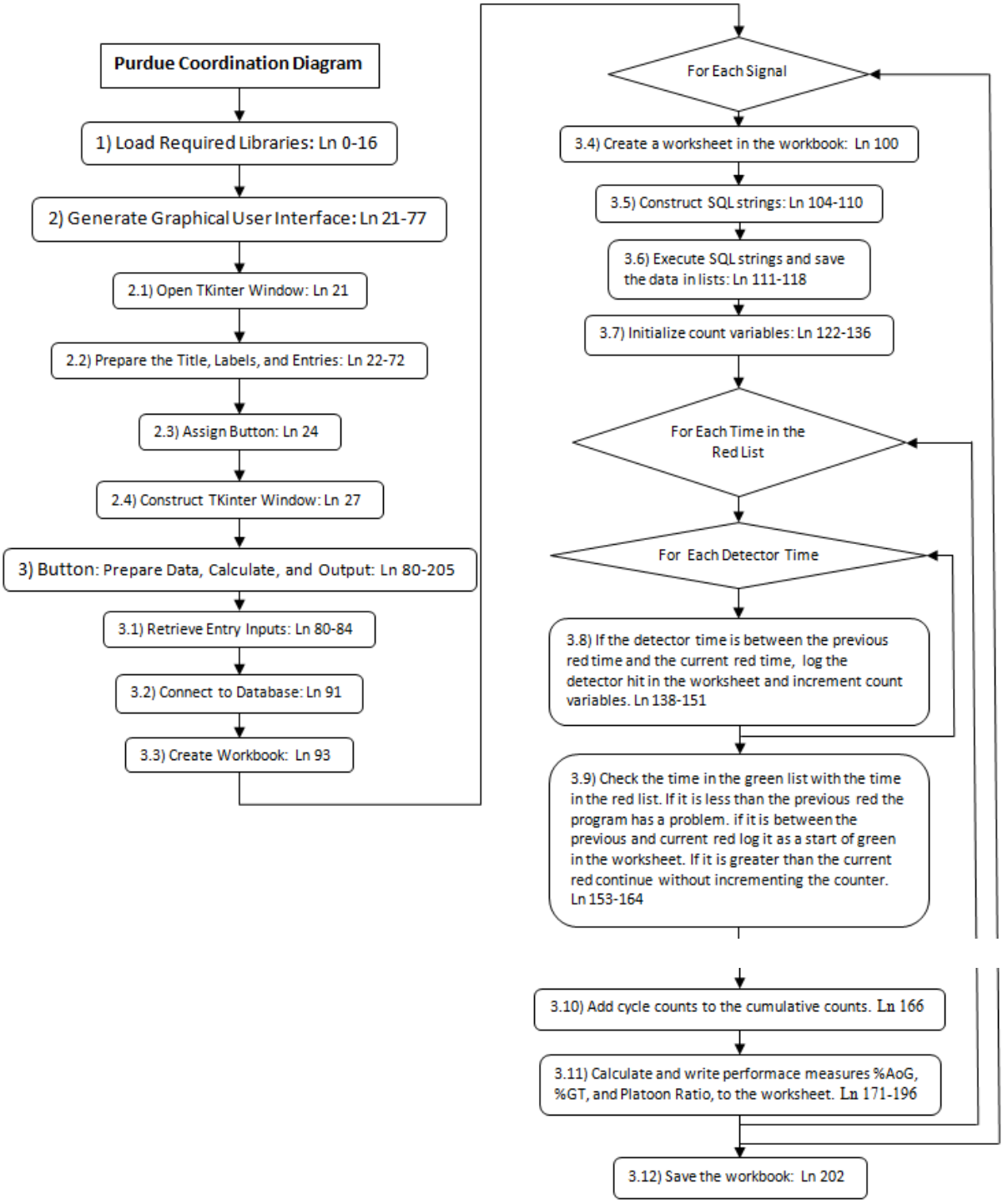


Figure 3-4. Purdue Coordination Diagram Flowchart

3.3.4 Coding Guidance for Anticipated Improvements

3.3.4.1 Incorporating Upstream Intersection Detection for Arrival Estimation

PCD oriented information is most useful when detection occurs upstream of the subject intersection's queue. In many cases, there are no detectors at this strategic location. Using arrival estimations from an upstream intersection is an easily implemented addition to the program. There will need to be an addition to the GUI's current entry fields, allowing the user to enter the upstream intersection information, which is the intersection number and the phase numbers that contribute to the subject intersection/phase arrivals. This program would then need to collect departure data which will be processed to create the list of arrival times. For example, the program would retrieve the departure times and add the travel time between intersections. The travel time could be based on probe data or simply defined as the distance divided by the estimated speed resulting in estimated arrival times. The program would then need refer to these estimated values instead of retrieving arrival times directly from queried detector results and it will produce the same output with the estimations.

3.3.4.2 Automating the Generation of Graphs in the Excel Output

Currently, the user needs to manually create the PCD from the output data that this program creates. The program can be modified to automate the generation of graphs in Excel using Visual Basic for Applications (VBA). The VBA program can either be run in Excel after the output file has been created or VBA can be executed by the Python program after the workbook has been saved.

3.4 Green Time Utilization

3.4.1 Introduction

This program analyzes high resolution data and output from an Excel file that contains both cyclic and overall performance measures. The primary performance measure calculated is the GTU.

However, detector hits are also counted, which can be used to measure the most basic performance measure, flow rates.

3.4.2 Interface

This interface, shown below in Figure 3-5, is very similar to the Purdue Coordination Diagram's interface. There are four entry fields: Signals, Phases, Detectors, and Table Name. They work the same as the PCD. The Signals field accepts comma-delimited values. The Phases field requires the same number of entries as the signals field. The intersection and phase values are considered as pairs. For instance, the first intersection-phase pair would be 1-2 and the second pair would be 1-6. The detectors field follows the same rules as they have in the previous interface but require a different set of detectors to be entered. For the best results, the PCD program uses advance detectors, while stop bar detectors would be best for the GTU measurement. As is shown below the detector field is too small for large entries. If more than one intersection phase pair is being analyzed it is recommended that the values be changed using the CustomDefaults.py file. The final entry is the name of the high resolution data table. The "Run Calculations" button will run the calculations and produce an Excel file with the calculated measures.

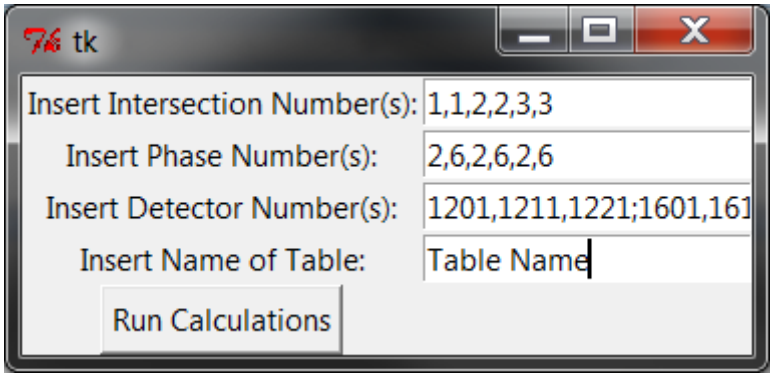


Figure 3-5. Green Time Utilization Interface

3.4.3 Flow Chart Narration

This discussion provides an overview for the computer program that extracts GTU information from high resolution controller data and uses the flowchart in Figure 3-6for illustration. The program starts by importing all the libraries required to function. These are all included with the scripts. The GUI is then constructed. There are 4 entry fields and a button. The button “Run Calculations” executes this program’s only function. As shown in Steps 3.1 to 3.13, this function collects the data from the database, processes it, and outputs the results into an Excel file.

The first step is to collect the information entered into the entry fields, Step 3.1. Next it enters a loop where it analyzes each intersection-phase pair. In the loop, it initializes the temporary count and summation variables that will be used, Step 3.5. Then it produces a query that selects all detector activation and deactivation times along with the start of red and start of green times. Each of these times also has a corresponding event. These values are arranged in one list which is analyzed in the next steps, 3.8-3.10. Each loop processes one event. If the event is a green light a few things are done. First the cycle counter is incremented up one. Then the number of hits in the previous cycle is appended to a list where they are stored until the reporting step. This is followed by the green light Boolean being set to true and the previous green time being logged. Last the temporary variable that holds the previous detector activation is overwritten with the start of green. This has to be done because the detector events are only analyzed if the light is green. Therefore, if the detector activates during the red phase, which is very likely, the program will recognize the detector as on immediately, at the same time as the beginning of green. If the detector is not active when the light turns green then this temporary variable will be overwritten with the actual activation time before the time the detector is on is calculated. There are also a few steps that are taken if the event is a red light. First the green light Boolean is set to false. Then, the green time is calculated and recorded, and if the detectors are active the time it has been active is

calculated and added to the sum. Similar to storing the start of green in the temporary detector, the end of the green must turn off any detectors and log their time towards the sum. When the event is not either of these two then the program checks to see if the light is currently green. If it is green then it will process the events accordingly. This is done because the program is not concerned with detector events while the light is red. The program then loops through the detectors to see which detector had a state change. Each of the detectors' states is stored in a list. If any detector is active, then the green is still being used and will be counted towards the utilized part of the phase's green time. When the phase's detection status changes states from off to on, the size of the gap is calculated and the activation is logged. When the phase's detection status changes from on to off the duration of the activation is calculated and reported. Once all the events have been analyzed, the performance measures are calculated and output to Excel.

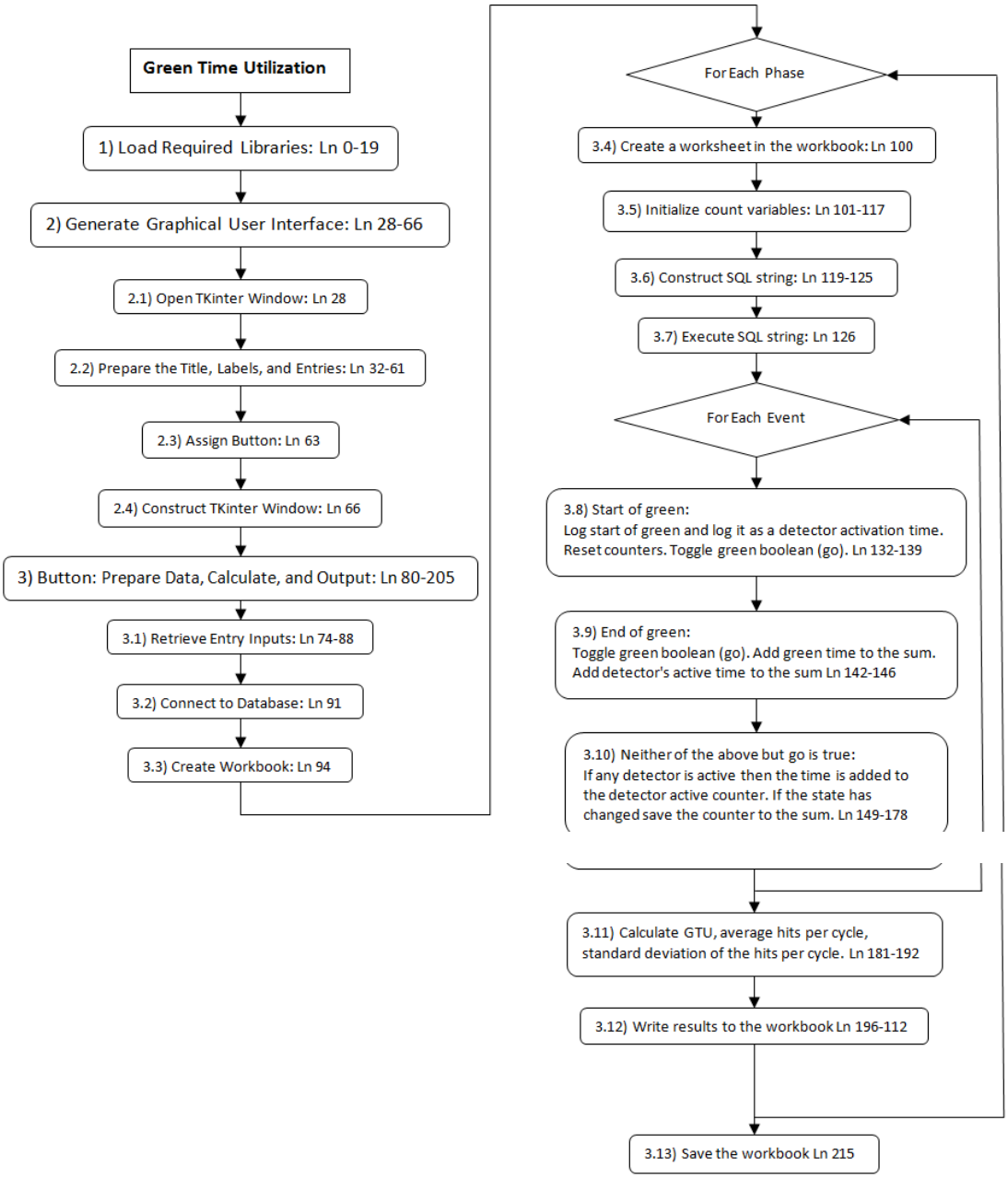


Figure 3-6. Green Time Utilization Flowchart

3.5 Phase Termination Analysis

3.5.1 Introduction

This program looks at the end-of-green events and uses the event code to prepare an Excel file. The gap-outs are separated from the max-outs and force-offs and both can be easily added to a graph for analysis.

3.5.2 Interface

The interface, shown below in Figure 3-7, is very similar to the GTU interface. It does not, however, require detector inputs of any kind, because it is only gathering reasons for phase termination. The phase entry is formatted similar to the detector input of the previous program. There can be multiple phase entries for each signal entry. Phase entries for the same signal are comma delimited and phase groups for each signal are separated by a semicolon, not a comma, see Figure 3-7.

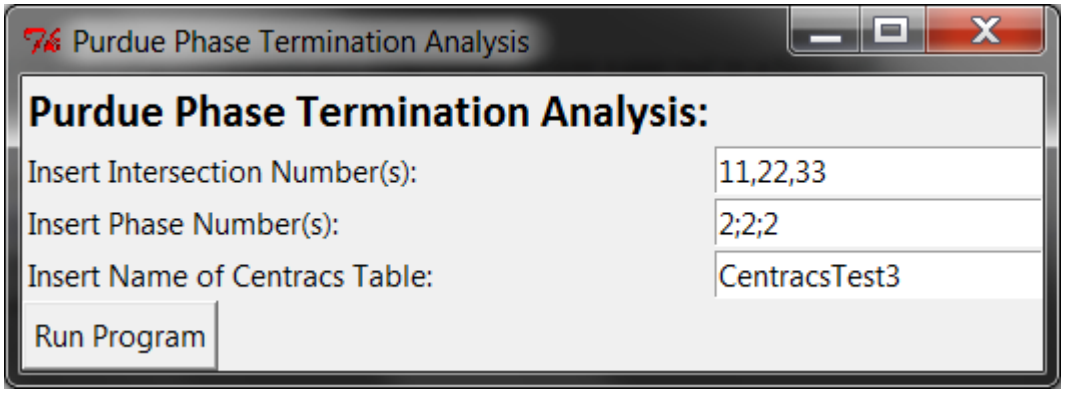


Figure 3-7. Phase Termination Interface

3.5.3 Flow Chart

This program has very simple calculations and, as shown in flowchart used for illustration, it is therefore significantly shorter than the others, see Figure 3-8. As with each of the programs, the first task is to load the required libraries. Then the program constructs the GUI. When the “Run Program” button is pressed the values of the entry fields are retrieved and used to create two queries. The queries collect the end of green times and gap-outs in one list, and force-offs/max-

outs in the other. For isolated intersection operations, force-offs should be considered max-outs. The lists retrieved from the queries are written to Excel workbook storing the output data. The workbook is then saved.

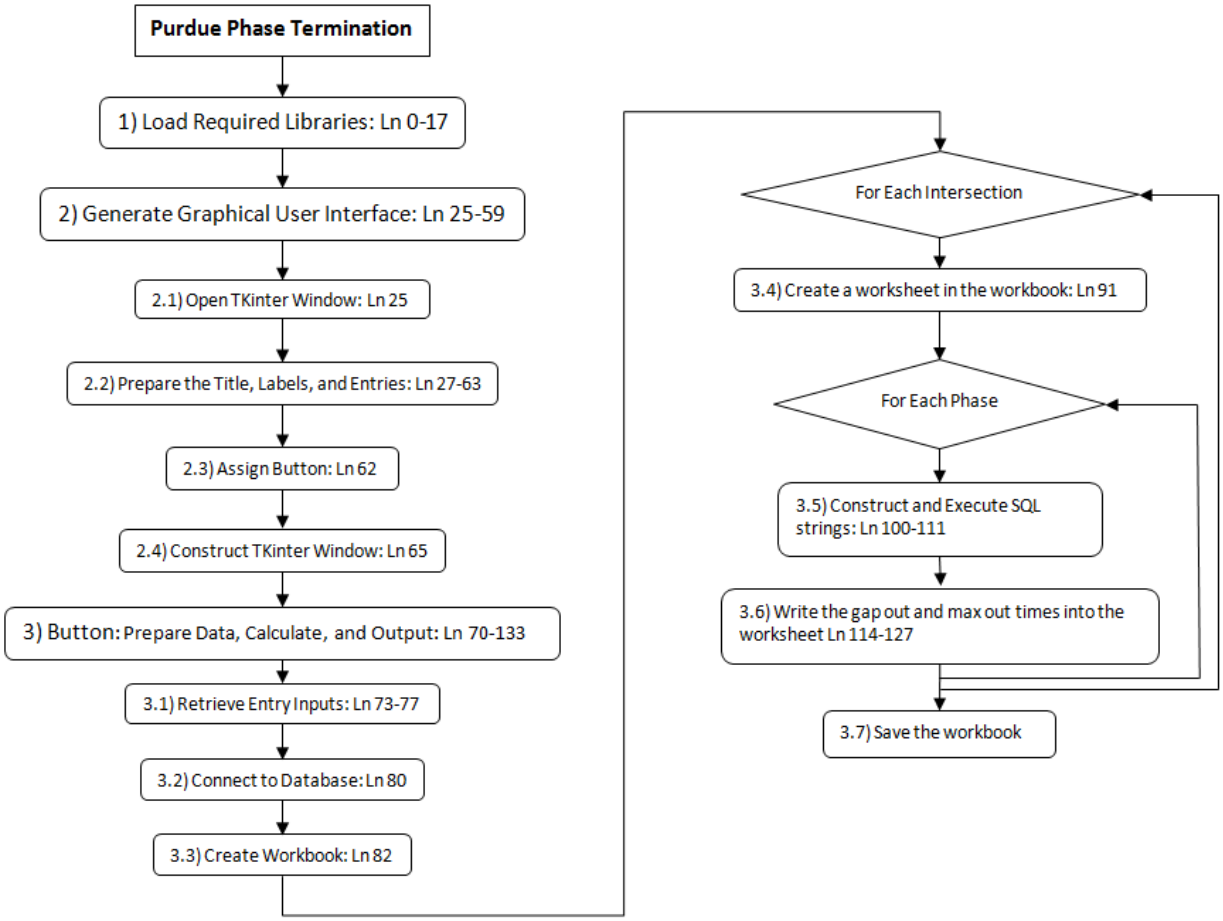


Figure 3-8. Phase Termination Flowchart

3.6 Delay and Queue Length Estimation

3.6.1 Introduction

This program estimates delay and queue length using high resolution data. The estimates are calculated using arrival and departure counts based on detector activations. This program requires lane-by-lane detection for both the arrival detection and the departure detection. Ideally, arrival

detection would be accomplished by advanced detectors located upstream of the queues and departure detection would occur through stop bar detectors.

3.6.2 Interface

Similar to the previously discussed programs, this interface has entry fields for the intersections, phases, arrival detectors, departure detectors, and the table name of the dataset, see Figure 3-9 below. The first notable difference is that there are two sets of detector inputs. This is because it is important to differentiate between the arrival detection and departure detection in the analysis. The intersection field requires that the intersection values be separated by commas. Similarly, the phase numbers must be separated by commas and must have the same number of entries as intersections. The detectors are broken into groups by semicolons which each refer to a phase. The detectors for a single phase are separated by commas. All of the information needed for delay at Phase 2 of Intersection 11 is indicated by dashed boxes in Figure 3-9. Queue length and delay could be extracted for other phases of the same intersection by repeating the intersection number, adding a different phase number, specifying the corresponding arrival and departure detector numbers.

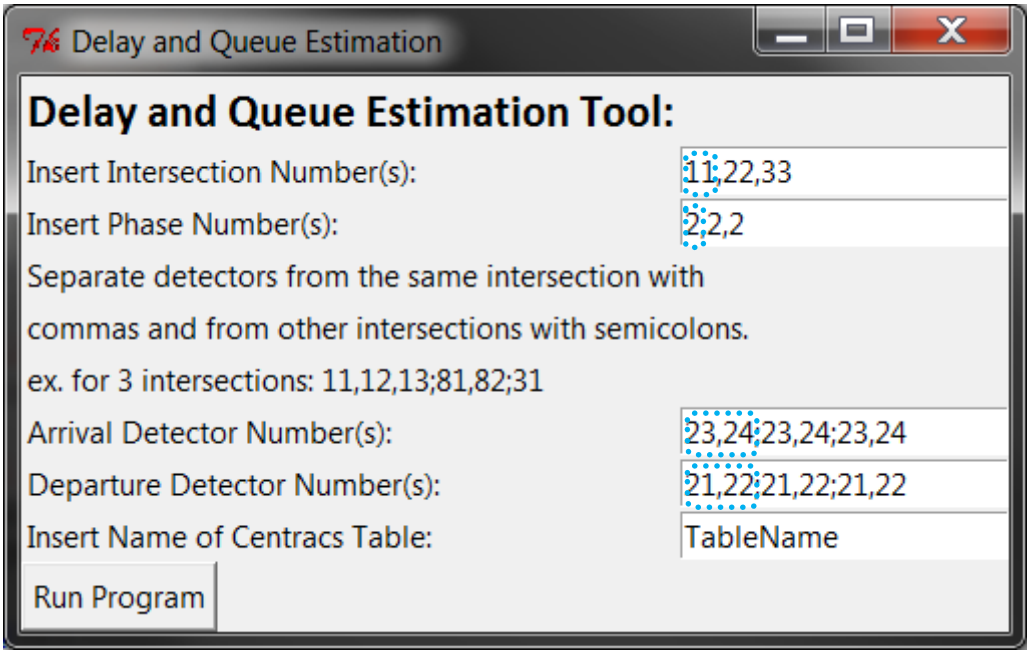


Figure 3-9. Delay and Queue Estimation Interface

3.6.3 Flow Chart Narration

The flow chart for this program is shown below in Figure 3-10. This program begins by loading the libraries it uses. Then it produces the GUI. When the "Run Program" button is pressed it executes the function. This function starts by retrieving the information in the entry fields. Then it opens a connection to the database, opens a workbook object, and begins looping through each of the intersection/phase combinations. For each combination, it creates a worksheet, retrieves the red times, retrieves the arrival times, and retrieves the departure times. Then it loops through each time step and checks for when different events occur (beginning of red, arrivals, and departures occur). If the event is the beginning of red then the queue length is recorded. If an arrival occurs the queue is incremented up one. If a departure occurs the queue is decremented down one. For each time step, the time, queue, and delay are recorded. Then the delay for the next time step is calculated. Once the loops have finished cycling through the lists, the workbook is saved.

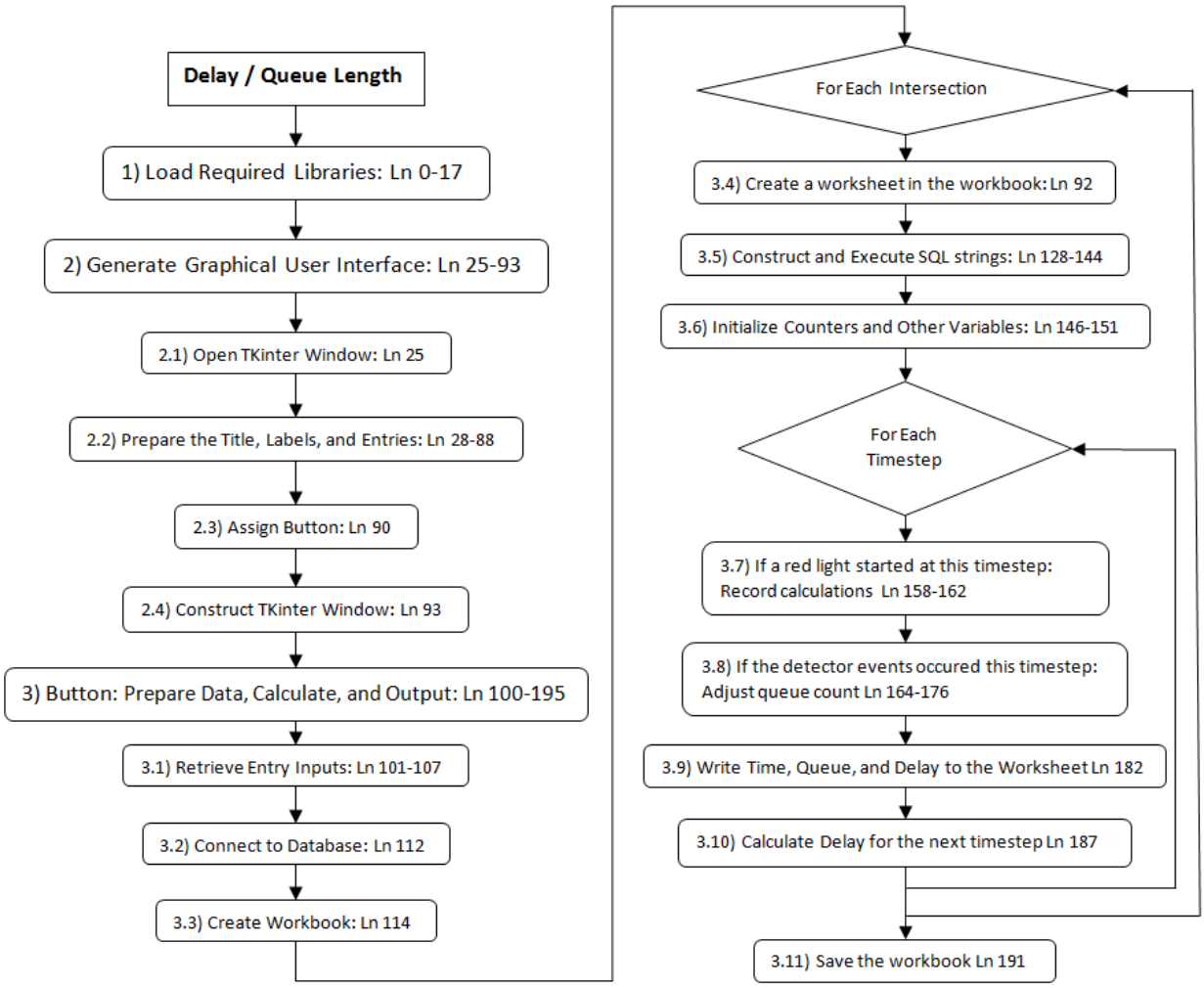


Figure 3-10. Delay/Queue length Flowchart

Chapter 4 TESTING

4.1 Overview

The output from the performance measure calculation tool was tested by directly comparing the measures produced by the tool with the hand calculated measures. All of the measures relied on the same VISSIM output files. The hand calculated measures, GTU, PCD, and PPTC, were produced using the detector output files (.ldp) and signal changes output file (.lsa) from VISSIM. The delay was compared to the calculated delay from VISSIM. Each measure was tested for accuracy over five cycles. Each of the tested measures was an exact match with the exception of the delay and queue length measures.

4.2 Purdue Coordination Diagram

The part of the tool that produced the PCD was tested by comparing the list of arrivals that were output to the detector output files. A PCD assembled in Excel using the detector output files was created to compare as well. These PCD's are shown below as Figure 4-1 and Figure 4-2.

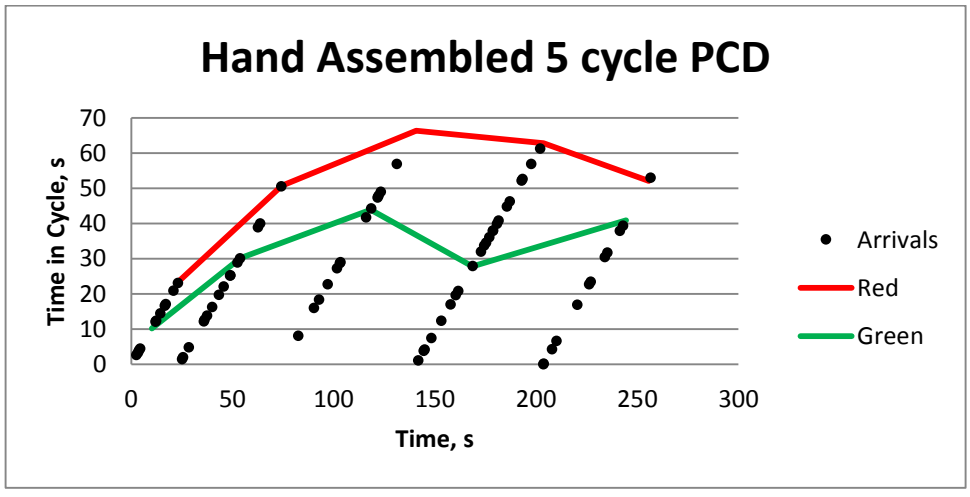


Figure 4-1. Hand Assembled PCD

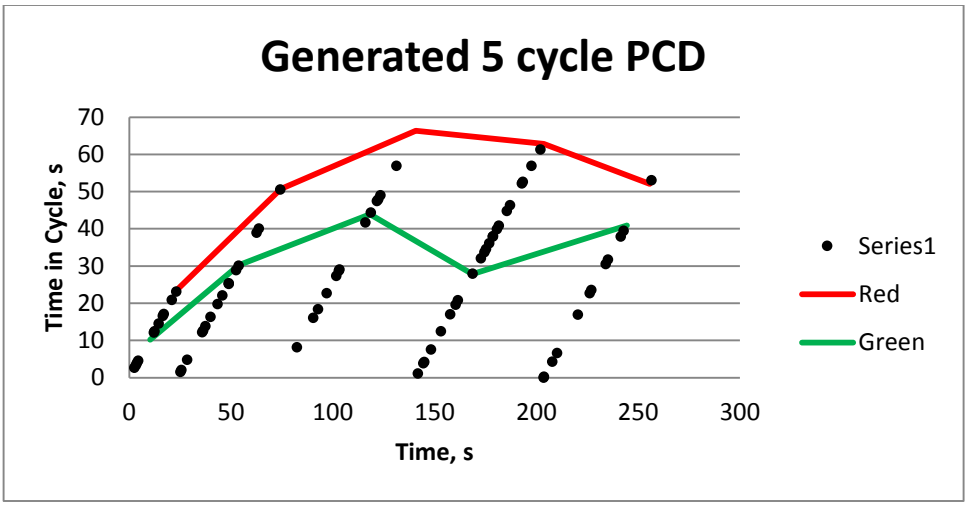


Figure 4-2. Tool Generated PCD

4.3 Green Time Utilization

The GTU performance measurement creation was the most difficult to calculate. The complications during calculation were primarily caused by detectors being active when the phase began. When a detector activated and deactivated during the phase the time that it was active is added to the total active time and is easily accounted for. When a detector is already active, at the start of the phase, the program wouldn't know it was active until the detector deactivated.

Table 4-1 below, shows the GTU that was produced by the program for two phases. The column titled "GTU 1" is the calculated GTU for the first five cycles of Phase 1 in Intersection 1. The values in the column to its right, labeled "Calculated" contain the hand calculated values. These values were identical to the ones produced by the program. The column "GTU 6" and its corresponding "Calculated" column were another test which included a phase with multiple lanes. The second test also produced identical results.

Table 4-1. Comparison of Hand Calculated GTU to Tool Calculated GTU

Cycle	GTU 1	Calculated	GTU 6	Calculated
1	0.27	0.27	0.21	0.21
2	0.33	0.33	0.32	0.32
3	0.14	0.14	0.22	0.22
4	0.34	0.34	0.24	0.24
5	0.36	0.36	0.38	0.38

4.4 Queue Length and Delay Estimation

Along with the other measures, queue length can be estimated using high-resolution data. This measure requires both advanced and stop bar detectors that are sensitive enough to detect individual vehicles. The queue length estimation is based on an input output model which assumes that every vehicle is detected, both entering and exiting the system.

To test the queue length estimation, VISSIM Data Collection Points (VDCP) were added to the system. These VDCPs provided the ground truth data on the traffic counts. Unlike the controller detectors, they have no influence on the intersection operations.

During an hour long simulation, manual inspection found, 15 errors in the controller detectors. The errors were caused by vehicles changing lanes over the advance detectors. These errors lead to a single vehicle being counted twice as it enters the system. The delay then has a bias of input vehicles because the vehicle was not double counted at the stop bar.

Accuracy was determined by calculating the delay with both the VDCP and controller detectors, with the VDCP being the ground truth. The results were assessed through visual inspection of the delay curves, shown below in Figure 4-3. The figure shows that the delay was accurate at the beginning of the simulation. The errors visibly separate the controller detector's delay from the VDCP's delay at cycle 5. Figure 4-3 below, shows the increasing errors in estimation as simulation

progressed. The controller detector's delay estimation is the line that continues to climb upward. On the other hand, an adjusted controller detector delay, which is explained in the next paragraph, and the ground truth calculation are steady near the 20 seconds/vehicle mark. Adjustments needed to be made to improve the calculations.

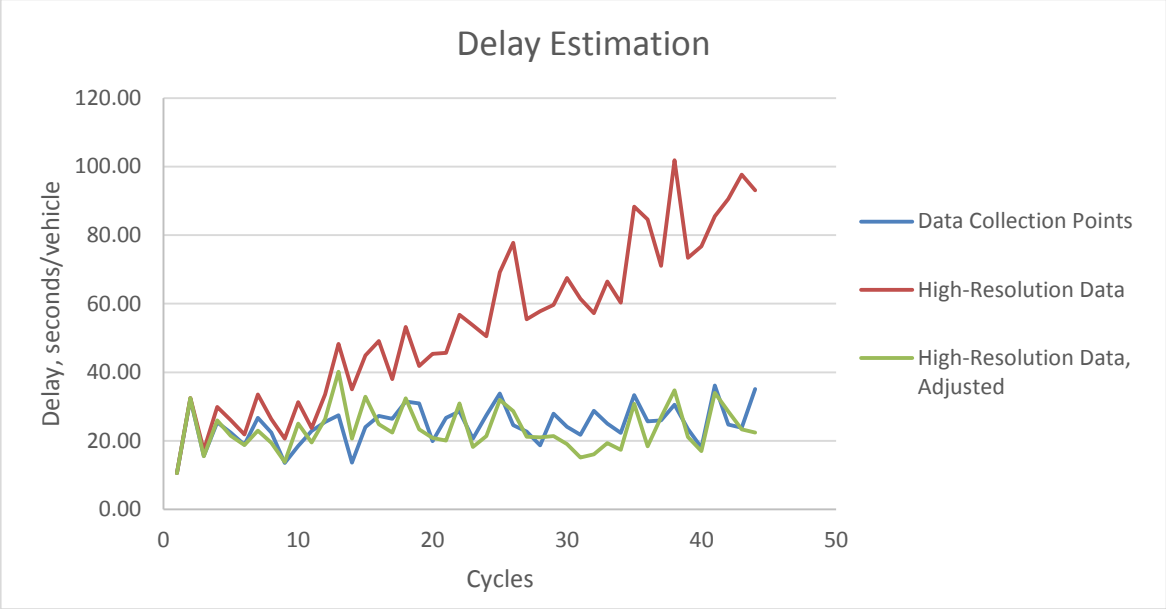


Figure 4-3. Cyclic delay of the different methods.

Since the cause for the errors was known, steps were taken to remove these imperfections from the dataset and retest the method with filtered controller detector data. Unfortunately, there was not a discernable pattern to the errors which would allow for an automated correction procedure.

A t-test determined the delay estimation method to be accurate when comparing the entire hour of data, with a t-statistic of 0.096. Even with the statistical confirmation it should be noted that the queue length estimation works most accurately with smaller datasets. Figure 4-4 and Figure 4-5 below, show the resulting cumulative vehicle diagrams for the ground truth data and the fixed high-resolution data. It is clear that the calculation starts out working well but as the slight inaccuracies of high-resolution data continue the accuracy decreases. Even in a plot that only shows the first five

minutes of the simulation, the high resolution estimations do not match the ground truth cumulative vehicles diagram.

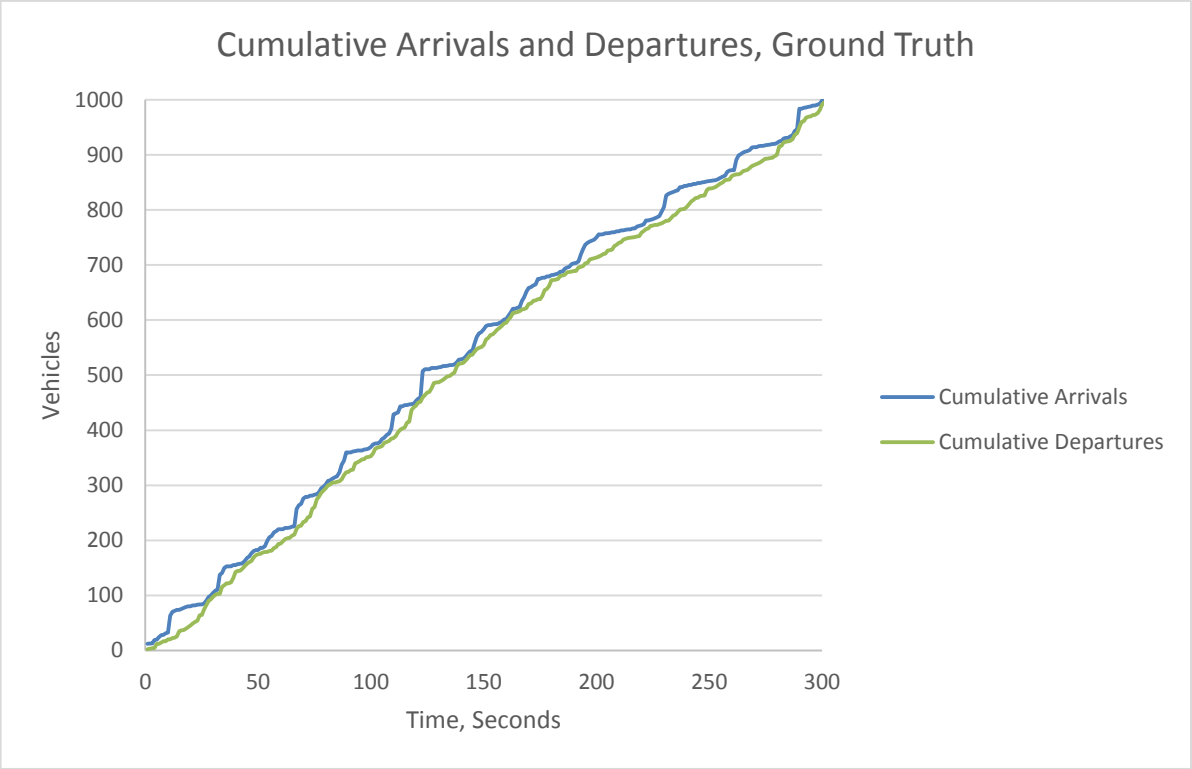


Figure 4-4. The ground truth calculation of the cumulative arrivals and departures for the first 300 seconds of the simulation.

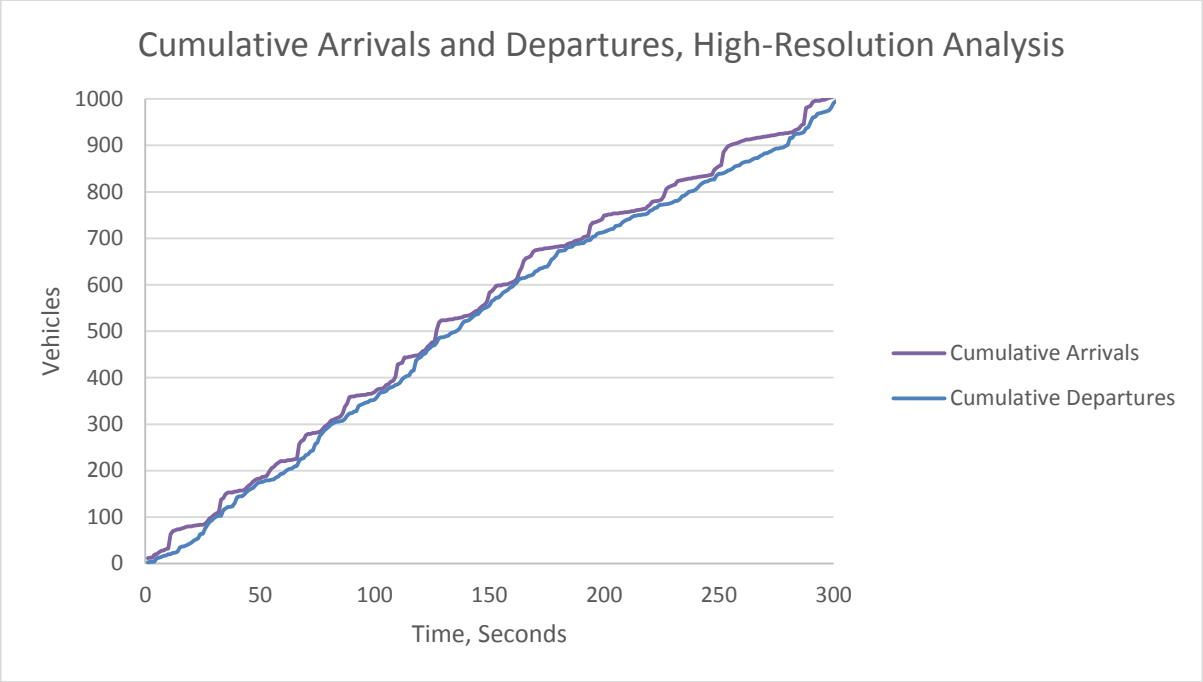


Figure 4-5. The controller detection delay calculation of the cumulative arrivals and departures for the first 300 seconds of the simulation.

The delay performance measure has potential in short analysis periods. However, using an algorithm to find and mitigate errors in the calculation would significantly improve the calculation. Also, instituting a process that can determine if there is a compounding error could significantly improve analysis as well as enable use outside of simulations. Including measures that correlate with delay would make this possible. For example, knowing whether or not a queue existed at the end of a phase by way of variations in the detector occupancy would eliminate additive count errors from one cycle to the next. This would essentially reduce the analysis period from the entire simulation to each individual cycle.

4.5 Split Failure Analysis

This is the most basic of the performance measures generated by the tool. The table below validates the results generated by the tool with hand checked phase terminations. Figure 4-6

below, shows the resulting chart from the validated tool results. The calculated measures were determined using the signal changes output file, shown in Table 4-2.

Table 4-2. Purdue Phase Termination Chart Method of Creation Comparison

Calculated Results		Program Results	
Time Stamp	Phase	Time Stamp	Phase
Gap-out		Gap-out	
5.2	1	5.2	1
109	1	109	1
235.7	1	235.7	1
Max-out		Max-out	
74.4	1	74.4	1
172	1	172	1
Gap-out		Gap-out	
23.7	6	23.7	6
74.4	6	74.4	6
140.8	6	140.8	6
255.7	6	255.7	6
Max-out		Max-out	
203.6	6	203.6	6

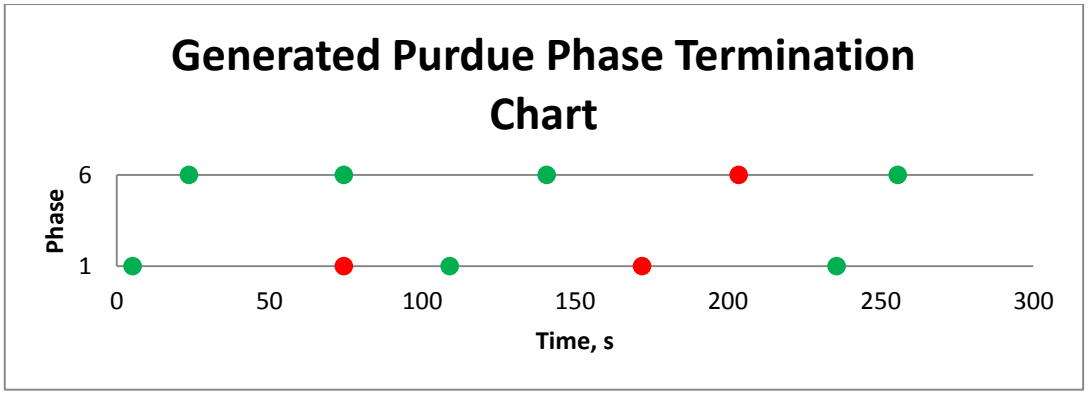


Figure 4-6. Purdue Phase Termination Chart

Chapter 5 CONCLUSIONS

This project produced a tool that generates high-resolution data from micro simulations without the use of hardware-in-the-loop simulation. Along with this functionality, the tool can calculate numerous performance measures using four different functions. These functions produce Excel files that facilitate the ability to analyze multiple measures at a time. This project has demonstrated and utilized the strengths of high resolution data. It has also shown that the data can be very limiting. The Purdue Coordination Diagram was very successful because the arrivals are accurately counted using the advance detectors. Along with the Purdue Coordination Diagram, the percent green time, percent arrivals on green, and platoon ratio were included so they could be plotted in the same figures. The Green Time Utilization was also very successful. While this measure is not definitive by its self, the tool produced greatly improves its usefulness by allowing this measure to be shown with others. The Purdue Phase Termination Chart was included in the project as well. Similar to the Green Time Utilization, this measure is best when used in conjunction with other measures. Delay and Queue length estimations were included in the project which had margins of error that varied significantly with the quality of the data and the length of the analysis period. This measure has the most potential as a standalone measure but requires more work. The results are promising when the data is perfect but imperfections in data can lead this measure off course. The focus of future work with this measure should start by error filtering the data being fed into this measure.

The tool was created using Python 2.7. The high-resolution data is generated by analyzing VISSIM output files. Future work will need to be done to enable the use of other micro simulation programs with this tool. Producing the high-resolution data with this tool requires setup but is a valuable alternative to using field data or hardware in the loop simulation. The alternative is to have the controllers linked to a system that produces a high-resolution database which is expensive

to buy and maintain. With this tool researchers will now be able to produce high-resolution data easily and efficiently. This will help accelerate the development of improved performance measures, which are becoming more valuable every day since multiple state agencies have begun collecting this data.

REFERENCES

Brennan, T.M., et al., (2011). “Visual Education Tools to Illustrate Coordinated System Operation” Transportation Research Board, National Research Council, Washington, DC.

Saito, Forbush, (2011). “AUTOMATED DELAY ESTIMATION AT SIGNALIZED INTERSECTIONS: PHASE I CONCEPT AND ALGORITHM DEVELOPMENT” Utah Department of Transportation Research and Development Division.

Smaglik, E. J., et a`. (2011). “Comparison of Alternative Real-Time Performance Measures for Measuring Signal Phase Utilization and Identifying Oversaturation” Transportation Research Board, National Research Council, Washington, DC.

UDOT, (2013). “http://udottraffic.utah.gov/signalperformancemetrics/”

Wang, Z., et al. (2014). “Estimating Queue Length at Signalized Intersections Using Multi-Source Data A Shockwave Theory Approach.pdf” Transportation Research Board, National Research Council, Washington, DC.