PERMANENT FREIGHT DATA COLLECTION INFRASTRUCTURE AND ARCHIVE SYSTEM

The Design, Integration and Testing of a Sensor Platform for Monitoring Truck Flows on Arterial Roads

Final Report for

METRO

Portland State UNIVERSITY

Prepared by

David S. Kim, J. David Porter, Seed Ghanbartehrani, Maryam Agahi
School of Mechanical, Industrial and Manufacturing Engineering

OSU Oregon State UNIVERSITY College of Engineering

June 2013

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

This report documents the results of the design, integration and testing of a sensor platform for monitoring truck flows on arterial roads based on data collected via inductive dual-loop detectors. The sensor platform, referred to as the Truck Count Unit (TCU), is composed of both commercial-off-the-shelf (COTS) and custom hardware components. The data collection and analysis software that processed the signals gathered by the inductive dual-loop detectors is based on an algorithm originally developed at the University of Washington. However, several modifications were needed to adapt this algorithm to the traffic flow conditions experienced by vehicles traveling on arterial roads.

The remainder of the report is organized as follows. Chapter 2 covers the assembly and configuration of the TCU. This chapter also includes instructions on how to install a fully configured the TCU in a loop detector cabinet. Chapter 3 describes the algorithm implemented in the TCU software to estimate vehicle length. Chapter 4 documents the results of testing the performance of the TCU at two different locations in Portland, Oregon. Finally, chapter 5 concludes this report with conclusions and recommendations.

2.0 ASSEMBLING AND CONFIGURING THE TRUCK COUNT UNIT

This section describes the hardware components needed to assemble a complete and functional TCU.

2.1. SINGLE BOARD COMPUTER

A single-board computer (SBC) is a complete computer built on a single circuit board and may include microprocessor(s), memory, input/output(I/O) and other features required of a functional computer. Figure 1 depicts an example of a basic SBC.

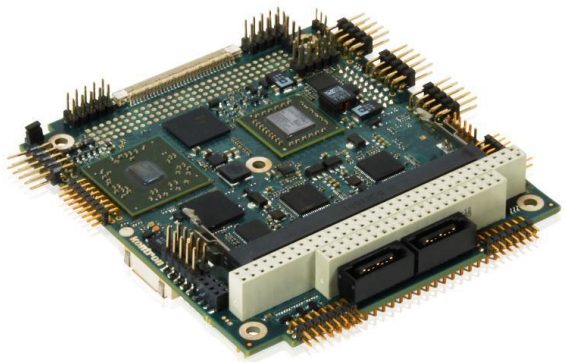

Figure 1. A Basic SBC

The following sections describe the *physical* and *operational* functional requirements expected of a SBC for the purposes of this project.

2.1.1. Physical Requirements

The physical requirements expected from a SBC that will serve as the foundation for a truck sensor are as follows:

- The SBC should be able to operate in extreme temperatures in the range of -10 °C to +50 °C.
- The SBC should not have a cooling fan, because cooling fans have mechanical parts that require maintenance. It is allowable to have heat sinks or heat dissipation plates.
- The materials used in the SBC's printed circuit board (PCB), plastic parts, passive and active elements should be durable. The use of solid-state capacitors is recommended.

2.1.2. Operational Requirements

The operational requirements expected from a SBC that will serve as the foundation for a truck sensor are as follows:

- The SBC should have different types of interfaces to communicate with other devices. This may include a serial port (e.g., RS232 or RS485), one or more USB ports and digital Input/Output (I/O) lines.
- The SBC should include a memory card slot to expand memory and easily load an operating system (OS) image on the card.
- The SBC should have the ability to run a high level OS (e.g., Linux, BSD, etc.) to simplify application development.
- The SBC should be robust and allow reliable operation 24 hours a day, 7 days a week. It should not crash after long hours of working continuously.
- The SBC should endure resetting due to power outage since there is no backup power present in control cabinets. Power interruption should not impact the SBC's hardware and/or software and should be able to recover from unexpected power outages.
- Preferably, there should not be an onboard battery. However, if there is one, it should be rechargeable.
- A software development kit (SDK) should be available with the SBC.
- The SBC should have a programmable watchdog time to monitor CPU operations, and to restart the CPU after a pre-determined, no response amount of time.
- The SBC should be provided with general purpose input/output (GPIO) lines to allow for direct input digital signals into the board. Changes in input signals should generate an interrupt to make real time processing of the signals possible.

2.1.3. Selected SBC

The process of searching for SBCs for the TCU was performed in early 2012. More than 200 SBCs from various companies were considered as candidates for the TCU. After carefully reviewing the physical and operational features of these options, two boards were chosen for final consideration. These SBC options and their most important features are shown in Table 1.

Table 1. List of selected SBCs

Model	Processor and RAM	Other Specifications	Distributor	Price
VDX-6354D	Vortex86DX 800MHz, 256 MB RAM	PC/104 VGA/LAN/Audio	Emac Inc.	$220
PCM-3343	Vortex86DX-1GHz, 256 MB RAM	PC/104/VGA w/dual LAN	Emac Inc.	$180

Since the two SBCs selected have almost the same features, both of them were purchased for initial testing. The SBC model VDX-6354D was more flexible with different operating systems; therefore, it was used in the development process of the TCU, where a copy of Debian Linux distribution along with a collection of development tools were required. The SBC model PCM-3343 comes with a faster processor at a slightly lower price, which makes it a better fit for TCUs; therefore, it was used to build the four prototypes currently installed.

It is important to note that there have been several newer, low-price SBCs introduced and made available recently to the market that can be used for the purposes of this project. Purchasing newer SBCs was not considered because the changes in project scope limited its application area to a few prototype units and porting already developed applications to a new hardware platform required more time and resources. However, the best substitute for the selected SBC (i.e., Emac's PCM-3343) is Raspberry Pi revision 2.0, which meets all the requirements and costs approximately $35.

2.2. TCU COMPONENT PARTS

Table 2 shows the component parts required to assemble a TCU. Parts names, quantities needed of each part, and the URLs to vendors' web sites are included. Figures 2 to 13 show pictures of each of the entries in Table.

Table 2. Components needed to assemble TCUs

No.	Name	Model/PN	Where to buy	QTY	Figure#
1	Team 2GB Compact Flash (CF) Flash Card	TG002G2NCFFX	http://www.newegg.com/Product/Product.aspx?Item=N82E16820313247	1	2
2	PCM-3343F board	1 GHz PC 104 SBC w Dual Lan CF 256 ram	http://www.emacinc.com/ordering.htm	1	3
3	Enclosure	HM1097-ND	http://www.digikey.com/product-detail/en/RM2055M/HM1097-ND/2094835	1	4
4	5V Power Adapter	T1038-P5RP-ND	http://www.digikey.com/product-detail/en/EPS050100-P5RP/T1038-P5RP-ND/2004025	1	5
5	5V LED	67-1062-ND	http://www.digikey.com/product-detail/en/SSL-LX3044GD-5V/67-1062-ND/270860	1	6
6	LED Holder	67-1330-ND	http://www.digikey.com/product-detail/en/SSH-LX3050/67-1330-ND/144804	1	7
7	Fuse Holder	708-1425-ND	http://www.digikey.com/product-detail/en/FX0462/708-1425-ND/1980782	1	8
8	Fuse	507-1277-ND	http://www.digikey.com/product-detail/en/5MT%201-R/507-1277-ND/1009049	1	9
9	Rocker Switch	SW339-ND	http://www.digikey.com/product-detail/en/GRS-4011-0024/SW339-ND/113625	1	10
10	Power Jack	SC1047-ND	http://www.digikey.com/product-detail/en/PC722A/SC1047-ND/109404	1	11
11	Board Stand off	1902DK-ND	http://www.digikey.com/product-detail/en/1902D/1902DK-ND/61872	6	12
12	The Interface Circuit	NA	Should be made/ ordered to made (schematic and part list are available)	1	13

Figure 2. Team 2GB Compact Flash (CF) Flash Card

Figure 3. PCM-3343F Board

Figure 4. Enclosure

Figure 5. 5V Power Adapter

Figure 6. 5V Power Adapter

Figure 7. LED Holder

Figure 8. Fuse Holder

Figure 9. Fuse

Figure 10. Rocker Switch

Figure 11. Power Jack

Figure 12. Board Stand Off

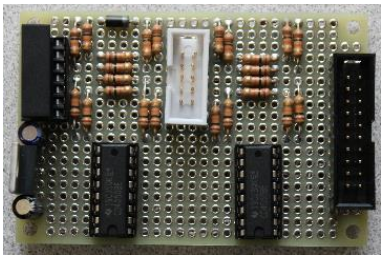

Figure 13. The Interface Circuit

2.2.1. The Interface Circuit

As explained in the requirements section, the SBC model PCM-3343 has a digital I/O data interface also known as a general purpose input/output (GPIO) interface which receives signals from the inductive dual-loop detector cards. Loop detector cards provide digital signals at their outputs, with a high voltage level varying anywhere between 12 and 24 volts (V) depending on the cabinet power supply. Since the GPIO ports on most SBCs can take up to 3.3V, a voltage conversion circuit is necessary to convert the voltage levels provided by the cabinet and also to provide protection for the SBC against changes in cabinet voltage.

Since no COTS solution was available for this purpose, a custom interface circuit was designed. Figure 14 shows the schematic of the interface circuit developed. A large size schematic of the interface circuit and a part list are provided in Appendix A.

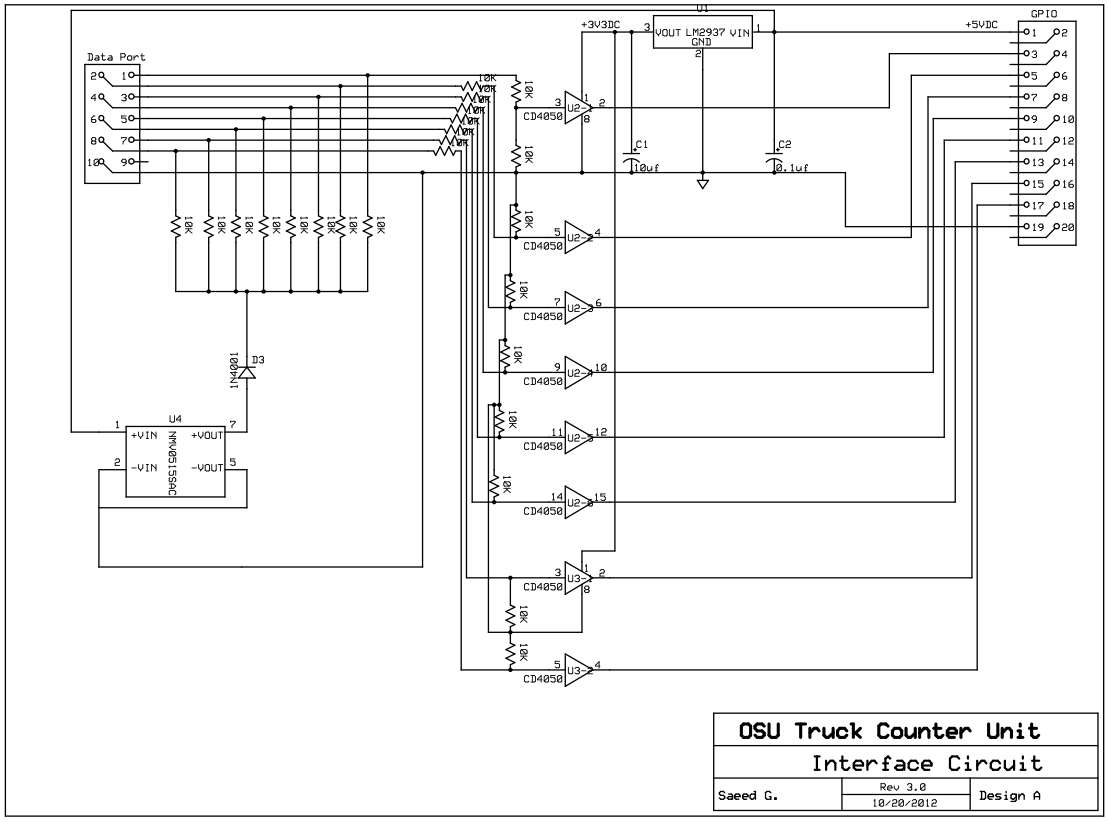

Figure 14. Electrical Diagram of the Interface Circuit

The design shown in Figure 14 is based on the popular CD4050 complementary metal–oxide–semiconductor (CMOS) to CMOS converter integrated circuit (IC). It comes with a 3.3V Voltage regulator (part# LM 2937) to provide reference voltage for the CD4050 and a 5V to 15V voltage converter (part# NMV0515SAC) to supply the loop detector card inputs, so the TCU can take data even if the loop detector card outputs are not connected to a traffic controller or a DC isolator.

The interface circuit has its own power sources and does not take excessive current from the inductive dual-loop detector card outputs or the traffic controller. Therefore, its operation has no effect on the performance of other devices connected to the loop detector card outputs. The inputs voltage levels are very flexible and can range between 10-36V. The SBC data inputs are isolated and voltage fluctuations in the cabinet power line should not cause any damage to the SBC.

A total of three interface circuits were assembled manually on prototyping PCBs for the TCUs. If larger quantities were needed, there are companies that can make the PCB and solder the parts. A picture of the finished prototype interface circuit and the two-connector cables needed are depicted Figure 15. The top cable shown in Figure 15 has a data port on one end and a connector on the other end (the latter attaches to the interface circuit). The cable with the two black connectors is used to connect the interface circuit into the SBC.

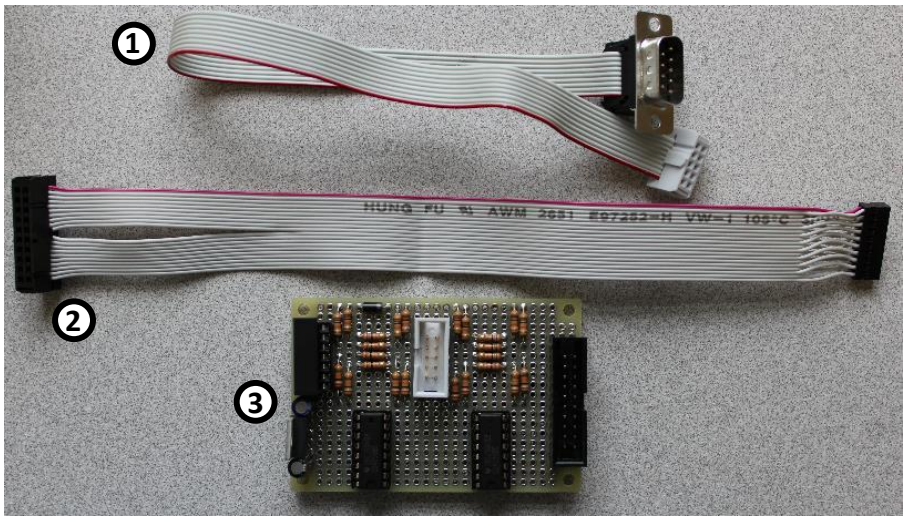

Figure 15. From Top to Bottom: (1) The Data Connector Cable, (2) The GPIO Connector Cable, and (3) The Interface Circuit

2.3. ASSEMBLING THE TCU

This section explains the process to assemble the TCU. First, an enclosure needs to be prepared to minimize the potential of the TCU being damaged by impact or other environmental conditions (e.g., dust). Next, several components need to be installed inside the enclosure. The CF card is installed next and finally all the components need to be connected together to produce a functioning TCU.

2.3.1. Preparing the Enclosure

It should be noted that a general-purpose enclosure was used with the prototype TCUs since the manufacturer of the SBC model PCM-3343 does not offer a compatible enclosure.

An acrylonitrile butadiene styrene (ABS) plastic enclosure seemed to best fit the purpose of the TCU designed for this project. ABS combines strength, rigidity, and toughness (Acrylonitrile butadiene styrene, 2013). An ABS plastic enclosure with detachable panels was chosen to make the customization process easier. All the holes and slots were drilled manually according to the diagram shown in Figure 16. A Microsoft® Visio file with all required information is also available in the attached CD that can be used for ordering custom enclosures.

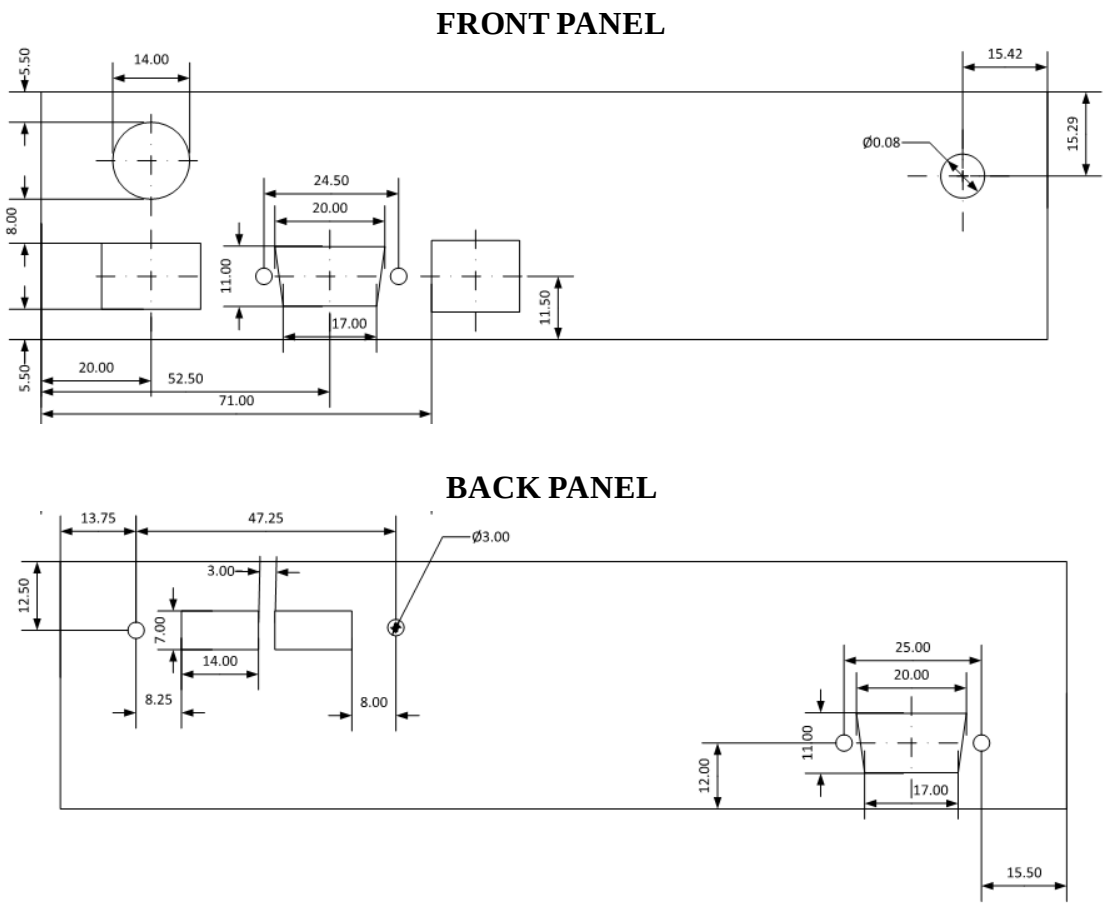

Figure 16. Drilling Guide for the Front and Rear Panels of the Enclosure

2.3.2. Installing Components in the Enclosure

The first step when installing components in the ABS plastic enclosure involves mounting the SBC model PCM-3343 and interface circuit on the bottom part of the enclosure. Both boards need to be secured with screws to plastic standoffs and then the bottom of the standoffs should be glued to the bottom inside surface of the ABS enclosure. Any industrial strength adhesive can be used for this purpose. The final position of the boards inside the ABS enclosure is depicted in Figure 17.

Figure 17. PCM-3343 SBC and Interface Circuit Mounted Inside the ABS Enclosure

Additional components should be mounted on the front and rear panels of the ABS enclosure. The power LED, the VGA port and USB ports should be mounted on the front panel, as depicted in Figure 18. Power jack, Ethernet port, serial data port, rocker switch and fuse holder should be mounted on the rear panel, as depicted in Figure 19. It should be noted that the VGA port, USB ports, and Ethernet connectors came with the SBC and therefore they are not listed in Table 2.

Figure 18. Components Installed on Front Panel

Figure 19. Components Installed on Rear Panel

After installing all the components on the front and rear panels of the ABS enclosure, they should be wired according to the diagram depicted in Figure 20.

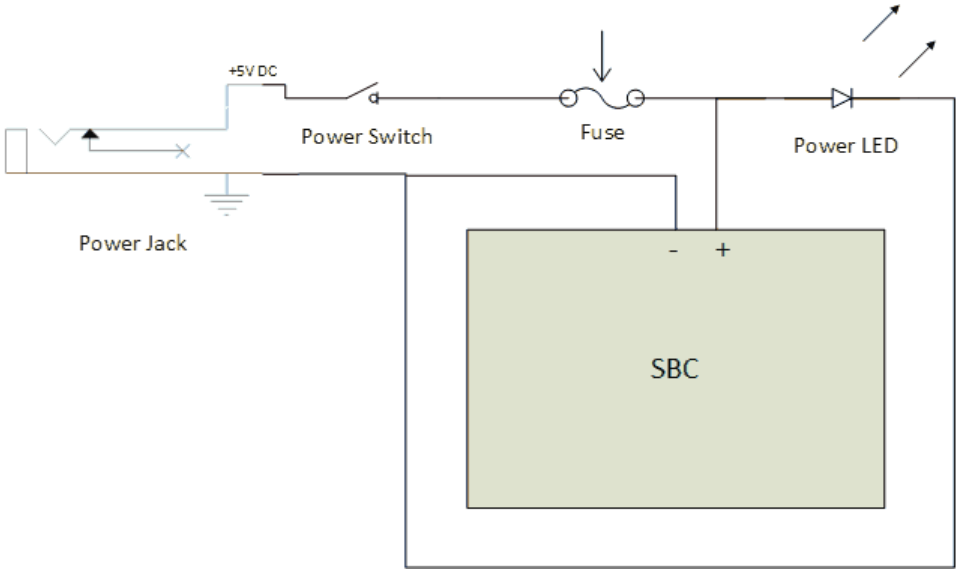

Figure 20: Enclosure Wiring Diagram

2.3.3. Installing the CF Card

The CF card needs to be prepared according to the instructions provided in section 2.4.1 to load all the software the TCU requires for operation. Then, the CF card should be inserted in its corresponding slot as shown in Figure 21.

Figure 21. Mounting the CF Card

2.3.4. Connecting Components Together

All the components installed on the front and rear panels of the ABS enclosure should be attached to their corresponding connectors on the SBC model PCM-3343, except for the serial data port which has a unique connector that should be attached to the interface circuit. It might be necessary to consult the manual of the SBC model PCM-3343 before connecting to the Ethernet and USB ports since they can be attached to various connectors on the SBC (and in different directions).

The GPIO connector, which is part of the interface circuit, should be attached to its corresponding connectors on the interface circuit and the SBC model PCM-3343. Figure 22 depicts all the components of the TCU properly attached to their corresponding connectors. It is critical to verify that all connectors have been plugged correctly (according to the SBC manual and schematics provided in attached CD) before closing the enclosure.

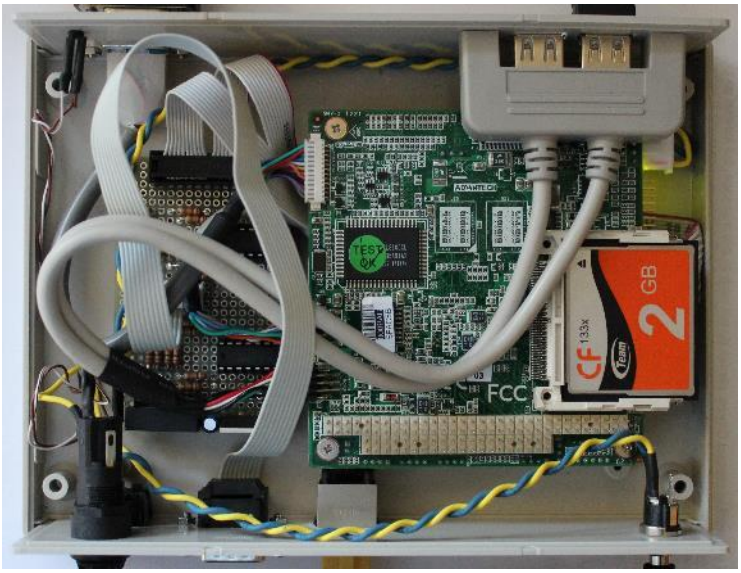

Figure 22. A Completely Assembled TCU

2.4. PROGRAMMING THE TCU

In this section, the procedures for programming and configuring the TCU before first use are presented.

2.4.1. Loading Software onto the CF Card

All the software needed for the TCU to function properly (i.e., the operating system, the data collection program, and the data archiving scripts) is stored in a two gigabyte (GB) CF card. This section outlines the steps required to format and install the software onto a CF card. It is assumed that a blank CFC will be “cloned” using a completely formatted

CF card. For the cloning procedure, a ready-to-go disk image is provided on the attached CD. The steps necessary to clone the CF card are as follows:

1. Insert the CF card into a card reader and connect the card reader to a computer running Linux. It is highly recommended that the Ubuntu Linux distribution is used in this process.

2. Before installing the image in the new and blank CF card, the CF card needs to be formatted using Gparted. In Ubuntu, Gparted can be found at *System* > *Administration* > *Gparted*. Once in Gparted, select the CF card (i.e., "/dev/sdb"). At this point in the process, it is important to verify that the correct CF card is selected and to verify its storage capacity. If there is any previous partition in the CF card, it must be erased. If there is no pre-existing partition, a new partition must be created. Right click on the partition and select "Unmount," as depicted in Figure 23, so Gparted can edit the partition table on the CF.

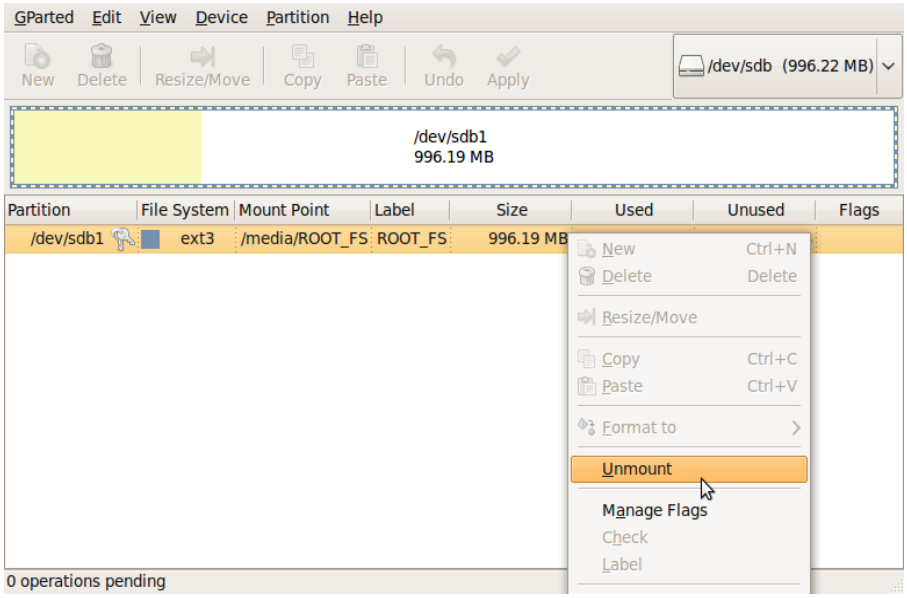

Figure 23. Unmounting the CF Card in Gparted

3. Left mouse click on the partition again to highlight it. The menu icon "Delete" will become active. Press the "Delete" button to delete the partition, as depicted in Figure 24.

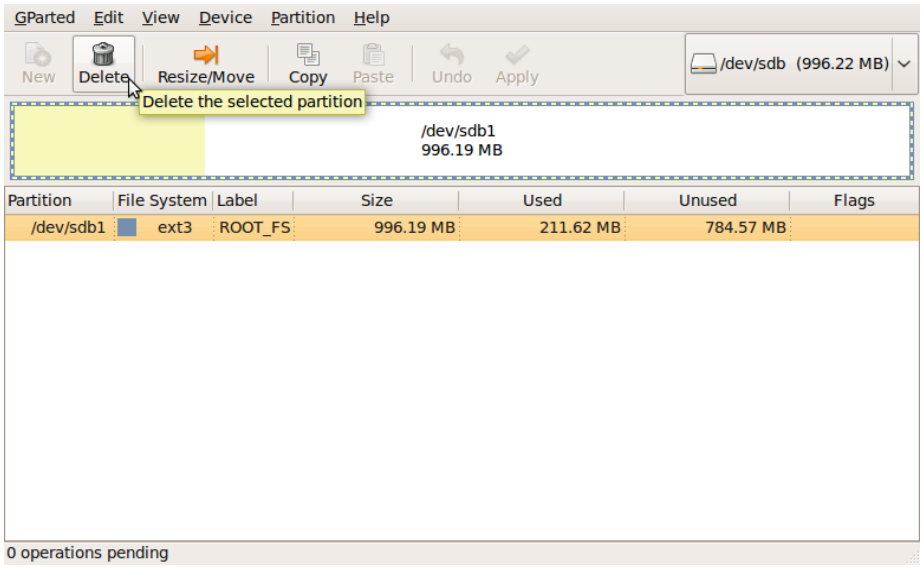

Figure 24. Deleting the CFC's Partition in Gparted

4. After clicking on the "Delete" icon, click on "Apply" to accept the changes. Confirm the selection when prompted (see Figure 25).

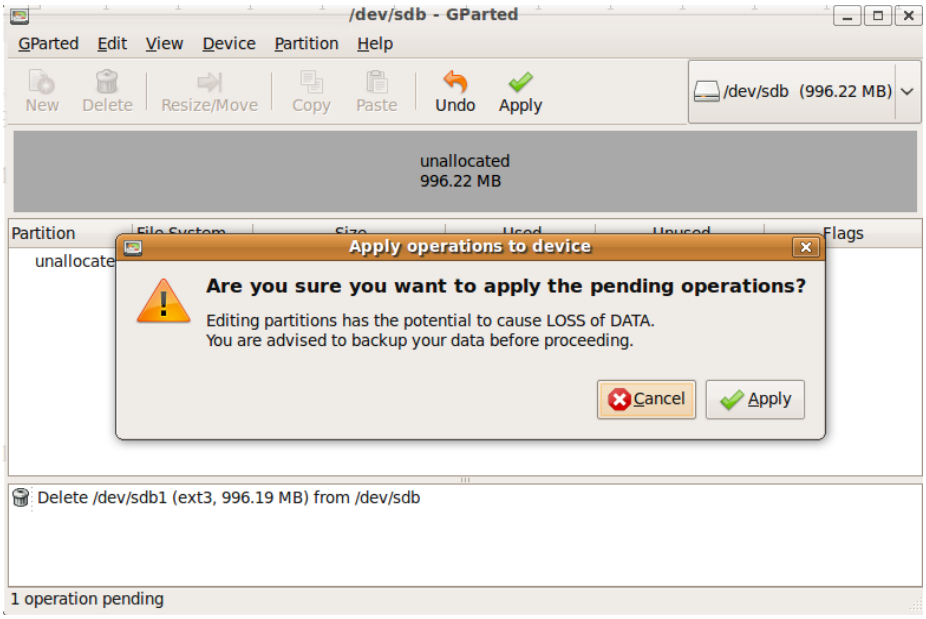

Figure 25. Deleting the CFC's Partition in Gparted (cont.)

5. After the configuration steps are completed in Gparted, a new partition must be created in the CF card. Highlight the partition and press the "New" button. A new window will appear. Select "ext3" from the drop down menu labeled "File System." Press the "Add" button to set the changes (see Figure 26) and this window will close. Click on "Apply" to create the partition.

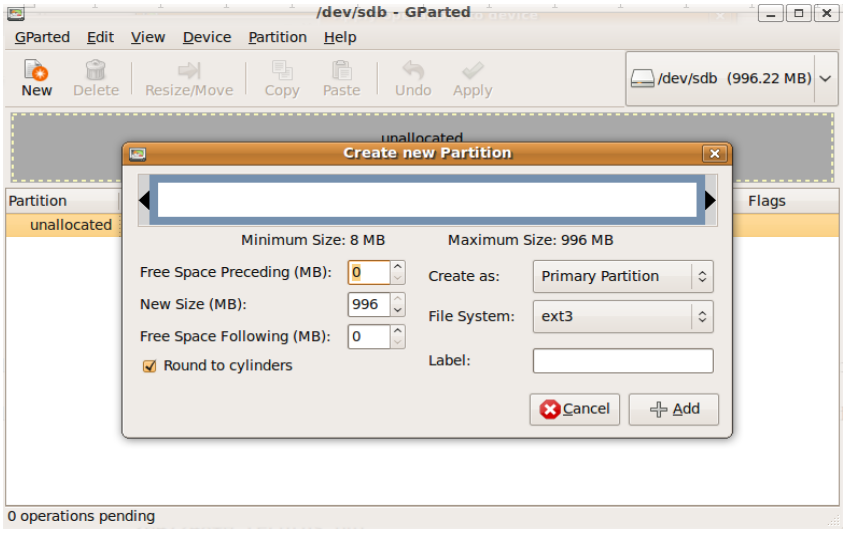

Figure 26. Creating a Partition in the CFC

6. Close Gparted and go to the Linux terminal window. In Ubuntu, the terminal window can be accessed via *Applications* > *Accessories* > *Terminal*. For this step, the user will need the provided a ready-to-go disk image. The user must copy the disk image into the folder "/tmp/." In Ubuntu, the easiest way to copy the file is to first copy it to the computer Desktop. Then, use the *move* command in Linux terminal to move the disk image to the "/tmp/" folder by typing "mv ~/Desktop/BootCF.img /tmp/" and then press <Enter>.

7. Run the command enclosed by the red rectangle shown at the top of Figure 27. Running this command may take several minutes and no feedback will be provided while the process is running. This command copies the image into the new CF card. In this example, it is assumed that the CF card image file is already copied to the folder "/tmp/" of the local computer. If the image file is stored in a different location, the command presented in this example must also be modified to read the disk image from a different location.

```
antar@yoda:~$ sudo dd if=/tmp/BootCF.img of=/dev/sdb
2047248+0 records in
2047248+0 records out
1048190976 bytes (1.0 GB) copied, 298.064 s, 3.5 MB/s
antar@yoda:~$
```

Figure 27. Copying Image to the CF Card

Once the output shown in Figure 27 is displayed, the CF card is ready to be inserted into the TCU. For the TCU to be completely functional, the appropriate network and timeserver parameters need to be set. Network parameters (e.g., IP address, subnet mask, etc.) are necessary so that the TCU can be accessed remotely. The timeserver parameters will allow the TCU to add accurate time stamps to the MAC address records it collects.

2.4.2. BIOS Settings

Before using the TCU, some changes in the basic input/output system (BIOS) settings are required. In order to access the BIOS settings page, the TCU should be connected to a monitor (via the VGA output) and a USB keyboard should be connected to one of its USB ports.

Press the <Del> key during startup to enter the BIOS CMOS Setup Utility. The BIOS main menu depicted in Figure 28 will appear on the screen. Select “Standard CMOS Features” and hit enter.

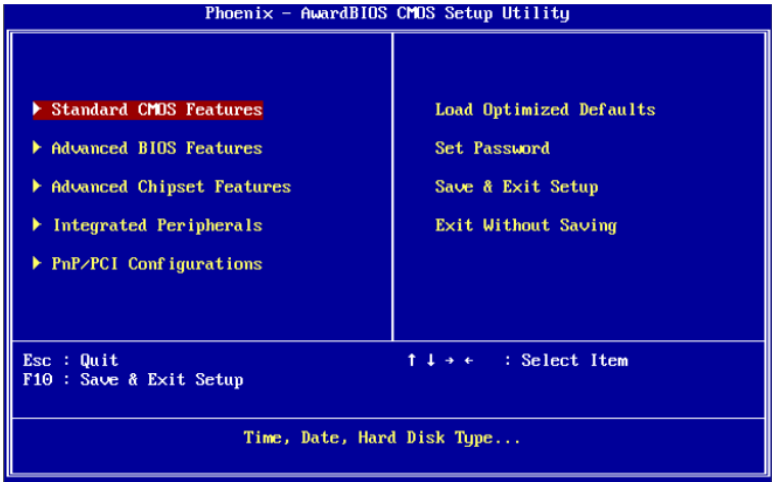

Figure 28. Main Menu for BIOS Settings

The screen depicted in Figure 29 should appear. Set the system date and system time in this screen. Then, return to the main menu by pressing the "Esc" key.

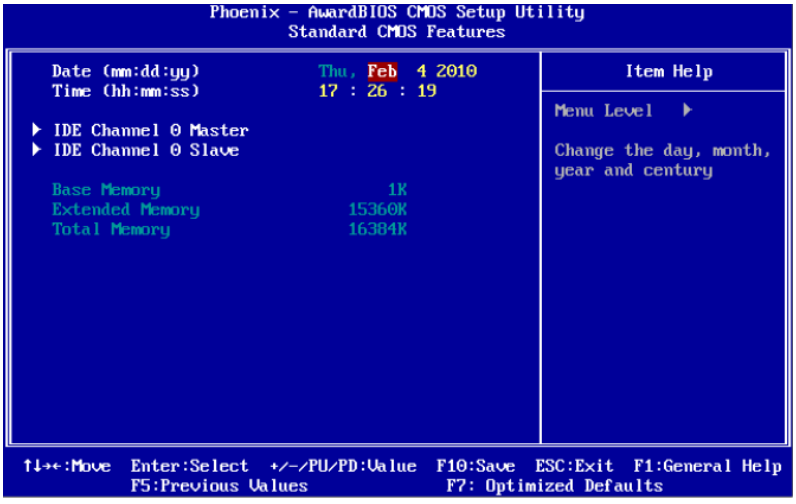

Figure 29. Standard CMOS Features

From the main menu screen, select the option “Advanced Chipset Features.” The screen depicted in Figure 30 should appear. In this screen, set “LAN2 Control” to "[Disabled]." Then, return to the main menu by pressing the "Esc" key.

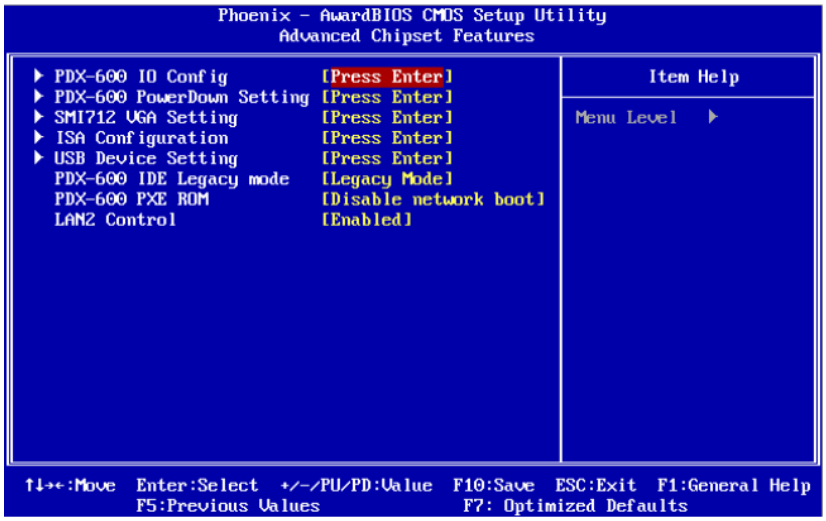

Figure 30. Advanced Chipset Features

From the main menu, choose “Save and Exit Setup” and type “Y” to confirm the changes as depicted in Figure 31. The system will automatically reboot. More details on BIOS settings can be found in chapter 3 of the user manual of the SBC model PCM-3343.

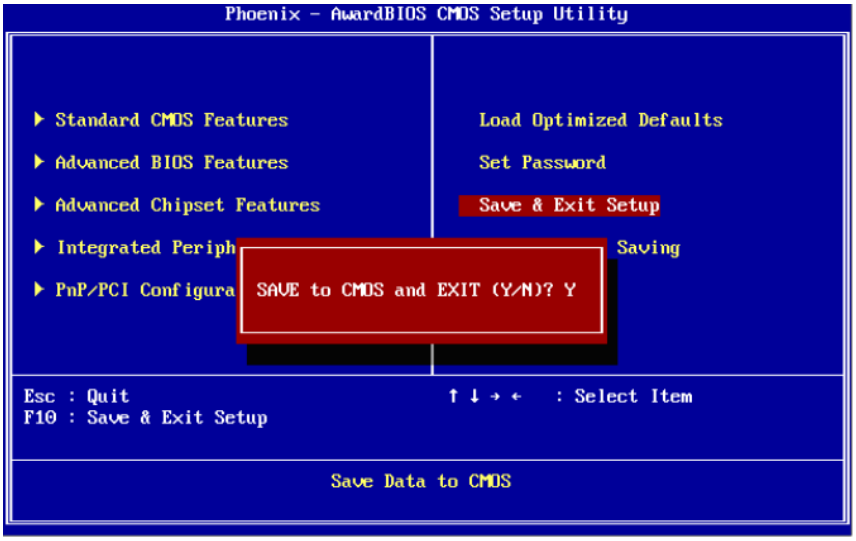

Figure 31. Saving data to CMOS

2.4.3. Other Settings

It takes 10-15 seconds for the TCU to boot up. In order to log into the system, “root” and “password” should be used as *username* and *password*.

2.4.3.1. Changing TCU Name and IP Address

In order to change the TCU name and IP configuration, the file system needs to be mounted as writable first. To do this, type */sbin/rw* at the prompt, as shown below, and then hit the <Enter> key.

```
[root@XLinux]:/xlinux # /sbin/rw
```

To change the TCU’s IP address and name, use a text editor (such as the *vi* text editor) to edit the file "/etc/init.d/rcS". To do this, type *vi /etc/init.d/rcS* and then hit the <Enter> key.

```
[root@XLinux]:/xlinux # vi /etc/init.d/rcS
```

Next, press the <Insert> key on the keyboard to enable editing mode. On line 2, change the “Linux_string” variable to the desired TCU name. In the example below, the TCU's name has been set to "OSU_Freemont."

```
Linux_string=OSU_Fremont
```

Still in editing mode, use the down arrow key to scroll down to line 41 and change the values for the IP address, network mask and the default gateway (gw), if required.

```
ifconfig eth0 10.107.1.156 netmask 255.255.255.0 up >/dev/null 2>&1
/sbin/route add default gw 10.107.1.1
```

After finishing all the necessary edits to "/etc/init.d/rcS", press the <Esc> key first, then press <Shift>+<:> and type "wq". Finally, press the <Enter> key to save the changes and to exit from the *vi* text editor. Reboot the TCU by typing reboot to have the changes take effect.

To exit without saving changes to "/etc/init.d/rcS", press <Shift>+<:> and type "q!".

2.4.4. Accessing the TCU Remotely

To access the TCU remotely or to log in without having a keyboard and monitor connected to the TCU, a secure shell (SSH) client like PuTTY can be used on computers running Windows. On Linux/Unix/Mac systems, the SSH command in a terminal can be used instead.

Network settings (e.g., IP, netmask, etc.) should be setup properly on the TCU and the computer connecting to the TCU should be connected to the same network the TCU is connected. PuTTY is a free SSH and Telnet client for Windows that can be downloaded from the following URL: http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe. If the above link does not work, navigate to the download page here: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html and try other mirrors.

PuTTY does not require installation. After download, double click on the executable file to run it. Figure 32 depicts the main screen of PuTTY.

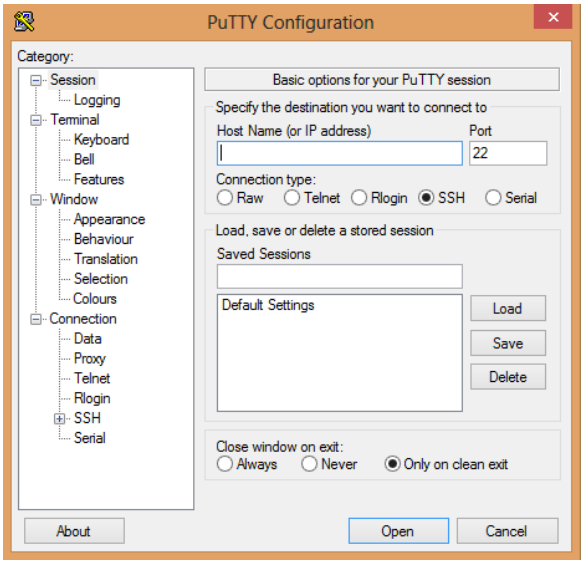

Figure 32. Main Screen of PuTTY

In the field labeled "Host Name", enter the TCU's IP address. Choose a name for the connection you are creating (e.g., TCU), enter it into the saved sessions field, and then click the "Save" button. The name "TCU" should be added to the list of saved sessions.

Select the connection just created (i.e., TCU), and click "Open". A new black window should appear after a few seconds with the prompt "login as". Type "root" and then press <Enter>. After a few seconds, a prompt will appear requesting a password. Enter "password" and then press <Enter>. Please note that when entering the password, no characters (not even asterisks) are displayed on the window and the cursor does not move at all. It may look like it is not capturing the password being typed, but it does. If an error is made while entering the password, press <Backspace> a few times to make sure you have erased all the characters you had entered and enter the password again.

2.5. INSTALLING THE TCU IN THE LOOP DETECTOR CABINET

Complete the following steps to install the TCU in the loop detector cabinet:

1. Place the TCU on a shelf in the loop detector cabinet. It is important that the TCU is placed where it does not interfere with the operation of other devices already installed in the cabinet. The TCU does not require ventilation and there are no air vents on the enclosure. However, the TCU should not block the air vents of other devices or placed where it may heat up due to the operation of other devices (e.g., on top of a traffic controller or behind a card rack).
2. Install the main harness of the TCU on the back of the input file (card rack) according to the wiring diagram instructions shown in Table 3. Table 3 provides all the wiring information for each pin of the header on the interface circuit and their corresponding positions on the serial DB9 connector on the back of the TCU. The wire colors noted on Table 3 correspond to the harness, which is provided with the prototype TCUs. If other DB9 harnesses are used, the color codes may be different and the connections have to be made according to the DB9 connector positions.

Table 3. Wiring diagram

Channel	Harness	DB9 pin Position	PCB	Header	PCB	DB9 pin Position	Harness	Channel
	NC	NC	NC	▪ ▪	GND	1	Black	Common
CH1_M	Green	6	P 0.0	▪ ▪	P 0.4	2	Brown	CH3_M
CH1_S	Blue	7	P 0.1	▪ ▪	P 0.5	3	Red	CH3_S
CH2_M	Purple	8	P 0.2	▪ ▪	P 0.6	4	Orange	CH4_M
CH2_S	Grey	9	P 0.3	▪ ▪	P 0.7	5	Yellow	CH4_S

Table 4 provides further information on the terminology used in Table 3. Each channel corresponds to a set of two loops. These channel numbers are also recorded in the TCU's output data file for every vehicle captured. It does not make any difference for the TCU if each channel is connected to a set of loops on northbound or southbound lanes, but it is important to connect upstream and downstream loops correctly according to the wiring diagram or no data will be captured.

Table 4. Description of conventions

Lane number	Channel	Loop
1	CH1_M	Loop 1, Upstream (Lead)
1	CH1_S	Loop 1, Downstream (Lag)
2	CH2_M	Loop 2, Upstream (Lead)
2	CH2_S	Loop 2, Downstream (Lag)
3	CH3_M	Loop 3, Upstream (Lead)
3	CH3_S	Loop 3, Downstream (Lag)
4	CH4_M	Loop 4, Upstream (Lead)
4	CH4_S	Loop 4, Downstream (Lag)

In order to keep the data captured by the TCU as accurate as possible, the output of the loop detector cards should be inputted into the TCU. Therefore, it is recommended to connect the data harness as close as possible to the loop detector card outputs (card rack). In case the signal source for the loops is located far away from the TCU's installation location and DC isolators are used to boost the signal, the TCU should be wired to the DC isolator inputs (i.e., where the DC isolator is supplied with the signal from the loop detector cards) as opposed to the DC isolator outputs (i.e., where the DC isolator provides amplified signal and the traffic controller and other devices are connected). The interface circuit is designed to be very sensitive and can pick signals as low as 8V. It also does not cause any voltage drops on the signal line, therefore it should not affect other devices, which are supplied with the same signal.

3. Plug the main harness connector to the data connector on the back of the unit and secure it.
4. Plug the power adapter to the 120V socket.
5. Plug the power cord into the power jack on the back of the TCU.
6. Switch the TCU on, and make sure the power LED lights up.

2.6. DOWNLOADING DATA FROM THE TCU

In this section, two methods are presented for downloading data from the TCU. The first method is suitable for onsite data retrieval or when the data needs to be downloaded manually. The second method is based on data retrieval scripts that are provided with the TCU and are useful when unattended remote data retrieval is required.

Please note that in order to use any of the presented methods, network settings (i.e., IP address and subnet mask) should be configured properly on the TCU. A computer with the Windows operating system needs to be connected to the same network that TCU is connected.

2.6.1. Data Archiving Scripts

The TCU is programmed to archive and compress the output data files in order to make automated file retrieval easier and faster. This section provides a short description on how these scripts operate.

The scripts are standard Linux shell scripts and are stored in the following paths:

- /sbin/archive
- /sbin/maintain

Note that the file system (/) is being mounted as read-only to protect the TCU against unexpected power failures. Therefore, in order to enable write operations on the disk, you

need to run "/sbin/rw" scripts (as described in the X-Linux manual) to re-mount it as writable.

The data collection software creates a text file every day to store that data for that day data in:

- /disk/data/*.txt (e.g., Dec04-12.txt)

The archiving script "/sbin/archive" runs every day five minutes after midnight (i.e., 00:05) and results in the following:

- The data file with the data collected from the prior day is compressed.
- The compressed file is appended (or copied, if no older archive is available) to the archive stored in /disk/data/archive and named as "Archived_`date +"%m-%d-%y"`.txt.gz" (e.g., Archived_Dec04-12.txt.gz).

On the 30th day of every month at 23:01, the maintenance script runs and result in the following:

- The old archive (ideally data from the previous two 2 months) stored in "/disk/data/archive/old" is removed.
- The current archive is moved to "/disk/data/archive/old".

Based on the above explanation, the best time to download data from the TCUs is on the 30th day of each month between 00:10 and 22:45 to avoid retrieving redundant data (e.g., copying files with overlapping data). So, if the files are downloaded during the mentioned time window (e.g., 30th day at 2:00), each file will contain data captured from the beginning of the 31st day of the previous month until the end of the 30th day of the current month. Please note that the time on the TCU is not being synchronized and, therefore, daylight saving changes have not been applied to them.

All these scripts are scheduled using "crontab" and in order to change the schedule, you need to edit "/crontab/root". The changes may not be effective unless the crontab service or the TCU is restarted.

2.6.2. Manual Data Retrieval

This section provides instructions to manually download data from the TCU data using WinSCP. WinSCP is a free file transfer protocol (FTP) client for Windows.

1. Download WinSCP from http://winscp.net/eng/download.php. After downloading and unzipping the WinSCP, go to step 2 to create a connection.

2. To create a connection, open WinSCP to see the main screen depicted in Figure 33. Click the “New” button on the top right of the main program window. A new window will appear (see Figure 34) and the TCU’s IP address along with username

and password must be entered. The default username is "root" the password is "password". Click "Save…" when done.

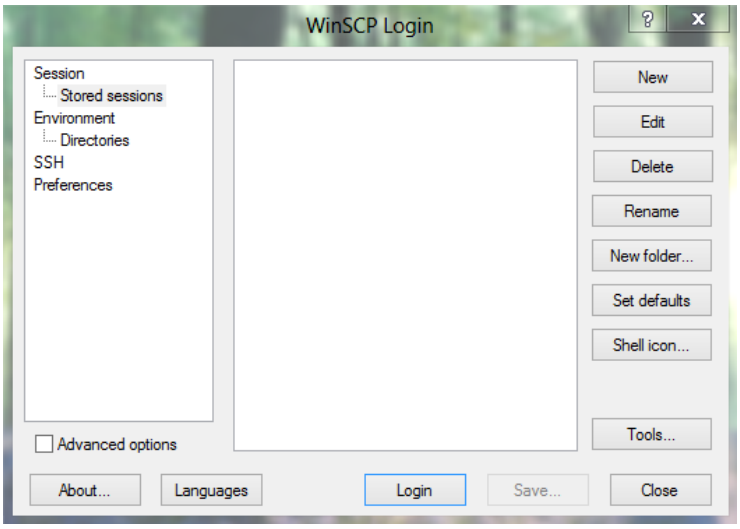

Figure 33. WinSCP Main Program Window

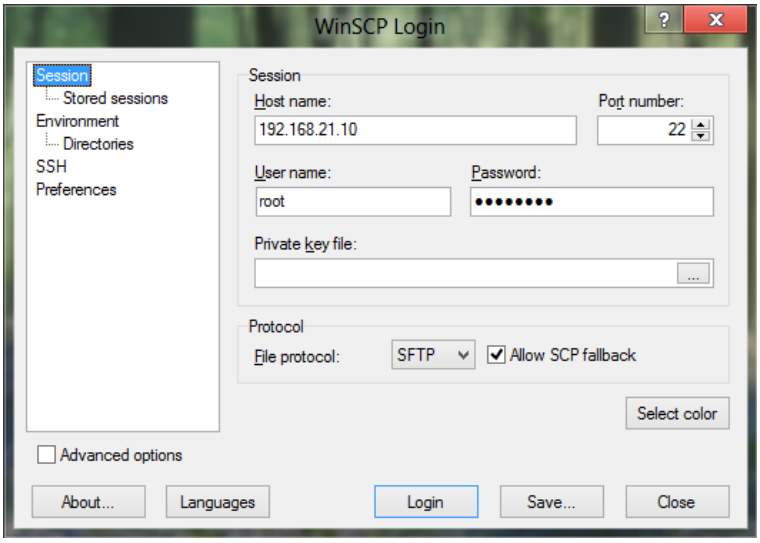

Figure 34. Parameter Values to Create a Connection in WinSCP

3. Select the connection just created and click “Login.”
4. To copy files from the TCU to the computer, complete the following steps:
   a. After a few seconds, a connection should be established. Accept any messages that show up during the logon process.
   b. A new window will appear with two frames. The right frame shows the files and directories available on the TCU. The left frame lists the files and directories on your computer.
   c. In the right frame, click on "disk" and then "data directories" to navigate to the subdirectory "/disk/data/archive".

d. Select all files in "/disk/data/archive" either by holding the <Ctrl> or the <Shift> keys and clicking on files.
e. In the left frame, navigate to the directory in the computer where the selected files from the TCU must be saved.
f. Drag the selected files from right frame to the left frame.
g. A window will appear to ask for confirmation to copy the files. Click "Copy" and wait until all files are downloaded.
h. Verify that all the files were copied to the computer and can be opened.

5. Close WinCSP to automatically terminate the connection with the TCU.

2.6.2. Automated Data Retrieval Using Script

Data can also be retrieved from the TCU with the use of an automatic retrieval script. The script is an MS DOS/Windows batch file that can be schedule to run automatically on a Windows-based computer and it is depicted in Figure 35.

```
rem Set variables below
rem manually run for the first time c:\82ndatBurnside>pscp -scp -unsafe -l
 root 10.107.1.155:/disk/data/archive/*.gz c:\82ndatBurnside and manually
 press y to accept a cached fingerprint
rem OSU box IP address:
set ip=10.107.1.155
rem OSU box root password:
set password=password
pscp -v -scp -unsafe -l root -pw %password%  %ip%:/disk/data/archive/*.gz
 c:\82ndatBurnside
pause
```

Figure 35. MS DOS/Windows Batch File for Automatic Data Retrieval from the TCU

The script requires a tool named *pscp* that can be downloaded at: http://www.nber.org/pscp.html. A folder should be created for storing the downloaded files (e.g., c:\82ndatBurnside) and an instance of pscp has to be copied there as well.

Data files stored in the TCU are labeled by date and therefore it is not necessary to rename them to avoid overwriting the old ones. The compressed files extracted from the TCU have the extension .gz and can be uncompressed using almost any popular archiving software such as the freeware "7zip". The script is shown below and a copy is also available in the attached CD.

3.0 DATA COLLECTION ALGORITHM AND SOFTWARE

In this chapter, the procedures implemented in the TCU software that are used to estimate vehicle length are described. A description and justification of the length estimates utilized is presented first. The remainder of the chapter documents the TCU code.

3.1 DUAL LOOP GEOMETRY

The basis for the algorithm used in this project to calculate the length and speed of a vehicle using signals collected from inductive dual-loops is an algorithm developed by researchers in the civil engineering department at the University of Washington (N. Nihan, Wang, & Zhang, 2005). The configuration of dual inductive loops within a single lane is shown in Figure 36. There is an upstream loop (M Loop) and a downstream loop (S Loop). There are wires that attach loops to a *loop detector card* installed in a traffic cabinet. A loop detector card detects changes in magnetic field caused by vehicles or any metal surface that enters the loops' area and translates them into on and off signals. When a vehicle drives over a loop, the loop detector card outputs a signal, whose length (in time units) ideally corresponds to the amount of time the vehicle has been present on each of the loops. In the notation used here, *TXON* refers to the moment that the signal on loop *X* changes to active (or ON level) and *TXOFF* is the moment that the *X* loop signal returns to inactive (or OFF) level, where *X = M or S.*

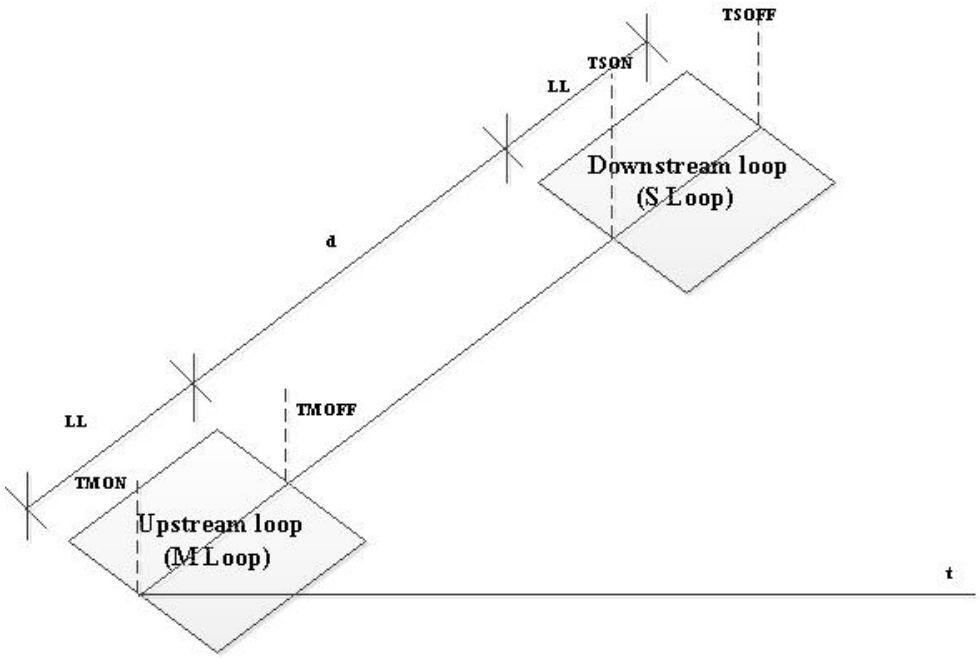

Figure 36. Dual Loop Detection System

The notation used in the description of the length estimates computed from inductive loop data is shown in Table 5.

Table 5. Notation used in the calculation of vehicle length estimates

Parameter	Description
TMOFF	Off-time for the M-Loop
TMON	On-time for the M Loop
TSOFF	Off-time for the S-Loop
TSON	On-time for the S Loop
d	Distance between the end of the M Loop and the beginning of the S Loop
LL	Length of a single loop (in the direction of vehicle travel), Assumed to be the same for both loops

The notation in Table 5 for loop ON and OFF times is applicable for a single vehicle and, assuming no data collection errors, *TMON* < *TMOFF* < *TSON* < *TSOFF*. Different vehicle speed estimates can be computed from these data. These speed estimates assume that a pairing of ON and OFF times that belong to the same vehicle is straightforward to identify as long as stop and go congestion is not encountered. Three different speed estimates can be computed, as follows:

$$S_{ON} = \text{Speed estimated from loop on} - \text{times} = \frac{LL+d}{TSON-TMON} \tag{3.1}$$

$$S_{OFF} = \text{Speed estimated from loop off} - \text{times} = \frac{LL+d}{TSOFF-TMOFF} \tag{3.2}$$

$$S_{AVG} = \frac{S_{ON}+S_{OFF}}{2} \tag{3.3}$$

Nihan, Wang, & Zhang (2005) use S_{AVG} to estimate vehicle lengths from the M loop ON time and the S loop ON time using the following formulas:

$$L_M = (TMON - TMOFF) \times S_{AVG} - LL \tag{3.4}$$

$$L_S = (TSON - TSOFF) \times S_{AVG} - LL \tag{3.5}$$

In their method, the vehicle length estimate is calculated by averaging equations 3.4 and 3.5, as follows:

$$L_V = \frac{L_M+L_S}{2} \tag{3.6}$$

The prior formulas assume that the loop detector data passes several data validity tests. However, many factors affect the performance of loop detector systems, so multiple data validity tests are conducted in the Nihan, Wang, & Zhang (2005) method. When a data

validity test is not passed, an error flag is generated along with an alternate speed estimate (S_{WSDOT}) and vehicle length estimate (L_{WSDOT}) that depends on the type of data error encountered. Either $L_{WSDOT} = L_M$ or $L_{WSDOT} = Ls$ will be used as length estimates when a data validity test is not passed.

These speed and length estimates assume free flow traffic with relatively constant speed for vehicles as they travel over the loops. On arterials, the free flow traffic assumption is not always met, thus four alternative length estimates were developed and tested that may perform better in an environment where acceleration and deceleration over the loops may occur as vehicles travel over the loops.

Length Estimate 1: Calculating the length based on the time a vehicle enters the first loop and leaves the second loop.

$$Length = (TSOFF - TMON) \times S_{avg} - (d + 2LL) \quad (3.7)$$

Length Estimate 2: Calculating the length based on the time a vehicle leaves the first loop and enters the second loop.

$$Length = (TSON - TMOFF) \times S_{OFF} - d \quad (3.8)$$

Length Estimate 3: Calculating the length based on the time a vehicle enters the first loop and leaves the first loop.

$$Length = (TMOFF - TMON) \times S_{ON} - LL \quad (3.9)$$

Length Estimate 4: Calculating the length based on the time a vehicle enters the second loop and leaves the second loop.

$$Length = (TSOFF - TSON) \times S_{ON} - LL \quad (3.10)$$

In addition to the estimators just described, Itekyala (2010) proposed an estimator called the *Constant Acceleration-based Vehicle Classification* (CAVC) model to estimate vehicle length in congested conditions. The development of the CAVC model assumed that a vehicle moves across the dual-loop detectors at variable speed resulting from constant acceleration. The length estimator is:

$$L_{CAVC} = \left[OnT_1 \times \left(\frac{d}{t}\right) + \left(\frac{OnT_1 - OnT_2) \times (OnT_1 - t)}{(OnT_1 + OnT_2)(OnT_2 - OnT_1 + t)}\right)\right] - LL \quad (3.11)$$

Where the notation not previously defined is:

OnT_1 - Total vehicle on-time over the M loop.
OnT_2 - Total vehicle on-time over the S loop.
t - Time taken for vehicle to travel from the beginning of M loop to the beginning of the S loop.

To evaluate the accuracy of the four new length estimators and L_{CAVC}, the length estimate generated by each estimator was compared to a known vehicle length under different scenarios in a deterministic simulation. Assuming a known loop length (*LL*), and the distance between loops (*d*), different scenarios were created by changing the vehicle speed when entering the first loop (V_0), the vehicle speed when leaving the second loop (V_1), and the known length of the vehicle. For each scenario, the values for *TSON*, *TSOFF*, *TMON*, and *TMOFF* could be calculated, from which the vehicle length estimates using the different estimators could be calculated. Scenarios were created by using different realistic vehicle lengths, and values for V_0 and V_1 that were varied in 5 MPH increments. Table 5 shows the results of 15 different scenarios.

A total of 149 scenarios were created. The scenarios were partitioned into three acceleration categories that subjectively appeared to have a single estimator that performed the best (i.e., having the largest number of scenarios where the estimator was the most accurate) with respect to the absolute difference from the actual vehicle length. The three acceleration categories are less than -20 ft/sec^2, between -20 ft/sec^2 and 20 ft/sec^2, and greater than 20 ft/sec^2. For scenarios with acceleration greater than 20 ft/sec^2, the new length estimator 3 performed the best. For acceleration less than -20 ft/sec^2, the new length estimator 4 performed the best. When acceleration is between 20 and -20 ft/sec^2, the L_V performed the best.

As a result of this testing, the Nihan, Wang, & Zhang (2005) method is used when the acceleration estimated from the loop detector data is between -20 ft/sec^2 and 20 ft/sec^2, estimator 3 is used when the acceleration estimated from the loop detector data is greater than 20 ft/sec^2, and length estimator 4 is used when the acceleration estimated from the loop detector data is less than -20 ft/sec^2. In some cases, the Nihan, Wang, & Zhang (2005) method will return L_S or L_M instead of L_V when the acceleration estimated from the loop detector data is between -20 ft/sec^2 and 20 ft/sec^2, but these cases will typically occur with large accelerations or decelerations combined with long vehicle length, and should be infrequent.

Table 6. The results of 15 scenarios to evaluate vehicle length estimators

	Scenario Parameters			New Length Estimators (feet)				Existing Length Estimators (feet)			
Run #	V_0 **(MPH)**	V_1 **(MPH)**	**Known Length (feet)**	**1**	**2**	**3**	**4**	L_V	L_M	L_S	L_{CAVC}
1	10	35	15	18.09	14.83	15.3	14.83	16.58	10.61	22.55	15.3
2	5	35	17	22.62	17.21	16.59	17.2	19.77	11.33	28.2	16.6
3	5	35	15	20.38	14.81	15.39	14.81	17.74	9.68	25.8	15.39
4	10	35	12	14.86	11.36	13.14	11.36	13.56	8.04	19.07	13.11
5	5	35	12	16.97	11.29	13.5	11.3	14.69	7.23	22.14	13.48
6	35	0	20	29.91	17.92	20.91	17.92	24.67	36.62	12.71	20.89
7	35	5	20	25.98	18.32	20.88	18.32	22.79	31.75	13.83	20.86
8	35	0	24	34.66	19.69	25.98	19.69	28.75	41.59	15.9	25.94
9	35	5	24	30.41	20.51	25.91	20.51	26.81	36.41	17.21	25.87
10	35	0	27	38.19	20.94	29.87	20.94	31.8	45.27	18.33	29.82
11	35	5	27	33.71	22.07	29.77	22.07	29.82	39.86	19.77	29.72
12	35	0	30	41.7	22.14	33.83	22.14	34.85	48.92	20.78	33.76
13	35	5	30	37	23.59	33.69	23.59	32.83	43.29	22.36	33.62
14	35	0	32	44.04	22.91	36.5	22.91	36.88	51.33	22.42	36.42
15	35	5	32	39.19	24.57	36.34	24.57	34.82	45.55	24.09	36.26

3.2 TCU SOFTWARE

In this section, the procedures implemented in the TCU data collection software are described. The data collection software was developed in the C programming language, and compiled using the GNU Compiler Collection in Linux. The binary executable files are included in the TCU disk image file and the full source is provided in the attached CD. Since there are C compilers (and cross compilers) for every platform (i.e., UNIX, Windows, Arm, etc.), the data collection software can be compiled and used in almost any platform.

3.2.1. Constants

As depicted in Figure 37, several constants are listed at the beginning of the data collection program. These constants are used for calculating speed and length, as well as other conditions such as error flags.

```
#define Dist 16  //Distance between the two loops in feet
#define Srate 681818.182 //ratio to convert ft/usec to MPH
#define LL 6 //Loop length in feet
#define Smin 5 //Min speed: 5 MPH
#define Smax 100 //Max speed: 100 MPH
#define Lmin 5 //Min vehicle length: 5 ft.
#define Lmax 110 //Max vehicle length: 110 ft.
const double Tonmin=75000;    //Min on time: 75 ms
const double Tonmax=1.583E6;  //Max on time: 1583 ms
const double Tofmin=170000;   //Min off time: 170 ms
const double Tofmax=0;        //Max off time: 0 ms
const double Temin=109000;    //Min elapsed time: 109 ms
const double Temax=2.182E6;   //Max elapsed time: 2182 ms
const double DTe=.10;         //Delta T elapsed threshold: 10%
const double DTo=.10;             //Delta T on threshold: 10%
char fpath[50]="//disk//data//";
const double wait=5E3; //polling interval in micro seconds
```

Figure 37. Constants used by the TCU's Data Collection Software

The constants for the distance between the loops (*Dist*) and the constant for the length of the loops (*LL*) are the most notable. These constants are set to the standard values used by the state of Oregon, i.e., 16 feet and 6 feet, respectively.

Additional constants include *fpath* which is set to where the output text file is stored, and *wait* which represents the rate at which the software checks the data port for signal changes. The latter constant is currently set to 5 milliseconds, which means that signals are collected from the inductive dual-loop detectors at a rate of 200 times per second.

The rest of the constants are used for error flag calculation and are explained in N. L. Nihan at al. (2005).

3.2.2. Procedures

The program consists of nine main procedures. Each procedure is presented and explained briefly in the next sections.

3.2.2.1. Procedure main()

The procedure *main()*, depicted in Figure 38, is the startup procedure for the TCU's data collection software. In this procedure, the TCU's data port is being refreshed in the defined intervals by calling the *scan()* procedure. If the data on the port has changed, then the *time()* procedure is called to record loop ON and OFF times. The time and data are processed by the procedure *calc()*.

```
void main(void)
{
int c = 255;
_Bool stop=0;
iopl(3);
/* set GPIO port0[7-0] as input mode */
outb(0x00,0x98);
while (!stop)
{
if (c!=scan())
{
c = calc(scan(),c,time());
}
else
usleep(wait);
}
iopl(1);
}
```

Figure 38. Procedure main()

3.2.2.2. Procedure scan()

The scan() procedure, depicted in Figure 39, scans the GPIO port 0 and returns data as an integer value.

```
int scan(void)
{
int c;
/* read data from GPIO port0 */
c = inb(0x78);
return c;
}
```

Figure 39. Procedure *scan()*

3.2.2.3. Procedure time()

The *time()* procedure, depicted in Figure 40, records the system time and returns a *timeval* structure. The time produced by this procedure is used in calculating loop on and off times.

```
struct timeval time(void)
{
struct timeval output;
gettimeofday(&output, NULL);
return output;
}
```

Figure 40. Procedure *time()*

3.2.2.4. Procedure calc()

The *calc()* procedure, depicted in Figure 41, scans and processes four channels. Each channel corresponds to a pair of loops (i.e., two bits of data), but more channels can be scanned and processed if necessary. The *calc()* procedure first compares the current state of the data port against its old state and determines whether or not the channel (or channels) have changed. If a change in state is detected, the calc() procedure calls the *process ()* procedure to process the changes.

```
int calc(int new, int old, struct timeval t)
{
int o,n;

//evaluating 1st channel
o = old & 3;
n = new & 3;
if (o!=n)
{
o = 4*o+n;
process(0,o,t);
}
//evaluating 2nd channel
o = (old & 12)/4;
n = (new & 12)/4;
if (o!=n)
{
o = 4*o+n;
process(1,o,t);
}
```

```
//evaluating 3rd channel
o = (old & 48)/16;
n = (new & 48)/16;
if (o!=n)
{
o = 4*o+n;
process(2,o,t);
}
//evaluating 4th channel
o = (old & 192)/64;
n = (new & 192)/64;
if (o!=n)
{
o = 4*o+n;
process(3,o,t);
}
return new;
}
```

Figure 41. Procedure *calc()*

3.2.2.5. Procedure process()

The procedure *process()*, depicted in Figure 42, processes the data collected from the inductive dual-loop detectors according to the type of change detected. In each case, the required data are calculated and in case the detected vehicle had finished traversing both loops (case 7), the *speed()* and the *length()* procedures are called to calculate speed and length, and the procedures *Eflag()* and *Cflag()* are called to calculate error flags. Then, all data are recorded in the data file and also displayed on the console. The data file is opened and closed every time a record needs to be added to minimize the chance of file corruption and data loss in case of power failure.

```
void process(int i, int input, struct timeval t)
{
FILE *fp;
char buf[30];
char fname[15];
float l;
switch (input)
{
    case 14:     //1st channel activated
          TS1[i] = ((t.tv_sec%1000)*1E6+t.tv_usec);
          TON1[i] = ((t.tv_sec%1000)*1E6+t.tv_usec);
          break;
    case 11:     //1st channel deactivated
          TS1[i] = ((t.tv_sec%1000)*1E6+t.tv_usec)-TS1[i];
          TOF1[i] = (t.tv_sec%1000)*1E6+t.tv_usec;
          break;
    case 8:             //2nd channel activated + 1st still active
          TM1[i] = (t.tv_sec%1000)*1E6+t.tv_usec;
          if (TON1[i]>0)
          {
          TON1[i] = ((t.tv_sec%1000)*1E6+t.tv_usec)-TON1[i];
          }
          break;
    case 1:
          TS1[i] = ((t.tv_sec%1000)*1E6+t.tv_usec)-TS1[i];
          break;
    case 9: //1st channel deactivated & 2nd channel activated (swap)
          TS1[i] = ((t.tv_sec%1000)*1E6+t.tv_usec)-TS1[i];
          TM1[i] = (t.tv_sec%1000)*1E6+t.tv_usec;
          TON1[i] = ((t.tv_sec%1000)*1E6+t.tv_usec)-TON1[i];
          TOF1[i] = (t.tv_sec%1000)*1E6+t.tv_usec;
          break;
    case 13: //2nd channel activated
          TM1[i] = (t.tv_sec%1000)*1E6+t.tv_usec;
          if (TON1[i]>0)
               {
          TON1[i] = ((t.tv_sec%1000)*1E6+t.tv_usec)-TON1[i];
          }
          break;
    case 7: //2nd channel deactivated
          if (TON1[i]>0)
          {
```

```
            TM1[i] = ((t.tv_sec%1000)*1E6+t.tv_usec)-TM1[i];
            TOF1[i] = ((t.tv_sec%1000)*1E6+t.tv_usec)-TOF1[i];
            strcpy(buf,ctime(&t));
            buf[strlen(buf)-1]='\0';
            Eflag (i);
            SP = speed(i);
            l = length(i,SP);
            Cflag(i,SP,l);
            printf ("%s Channel %d Speed %3.2f MPH Length(i) %3.2f VS l
%3.2f ft. Error %d TS=%1.0f TM=%1.0f TON=%1.0f
TOff=%1.0f\n",buf,i+1,SP,length(i,SP),l,flag[i], TS1[i], TM1[i], TON1[i],
TOF1[i]);
            strftime(fname, 15, "%b%d-%y.txt", localtime(&t));
            sprintf (fpath+14, fname);
            fp = fopen(fpath, "a"); //opening the file
            fprintf(fp,"%s Channel %d Speed %3.2f MPH Length %3.2f ft.
Error %d TS=%1.0f TM=%1.0f TON=%1.0f
TOff=%1.0f\n",buf,i+1,SP,length(i,SP),flag[i], TS1[i], TM1[i], TON1[i],
TOF1[i]);
            fclose(fp); //closing the file
            TS1[i] = 0;
            TM1[i] = 0;
            TON1[i] = 0;
            TOF1[i] = 0;
            }
            else
                  {
                  TS1[i] = 0;
                  TM1[i] = 0;
                  TON1[i] = 0;
                  TOF1[i] = 0;
                  }
            break;

      default:     //none
            TS1[i] = 0;
               TM1[i] = 0;
               TON1[i] = 0;
            TOF1[i] = 0;
            break;
   }
   }
```

Figure 42. Procedure *process()*

3.2.2.6. Procedure speed()

The procedure *speed()*, depicted in Figure 43, calculates a the speed of a vehicle based on the possible inconsistencies detected in the data captured from both inductive dual-loop detectors. This procedure was developed by Nancy L. Nihan, Wang, Cheevarunothai, & Northwest (2006). In case no error is detected, the speed is calculated based on an average of the ON and OFF times of each loop.

```
   float speed(int i)
   {
   float son,sof;
   double ton;
   int f;
   if ((flag[i] & 124)==0)
   {
       son = (float)(((Dist+LL)/TON1[i])*Srate);
       sof = (float)(((Dist+LL)/TOF1[i])*Srate);
       return ((son+sof)/2);
   }
   else if ((flag[i] & 124)==64)
   {
       if (SP>0)
       {
       ton = (((Srate*(Dist+LL))/SP)+0.5);
       if (fabs(TON1[i]-ton)<fabs(TOF1[i]-ton))
       {
              son = (float)(((Dist+LL)/TON1[i])*Srate);
              sof = (float)(((Dist+LL)/ton)*Srate);
              return ((son+sof)/2);
       }
       else
       {
              sof = (float)(((Dist+LL)/TOF1[i])*Srate);
              son = (float)(((Dist+LL)/ton)*Srate);
              return ((son+sof)/2);
       }
       }
       else
       {
       son = (float)(((Dist+LL)/TON1[i])*Srate);
       sof = (float)(((Dist+LL)/TOF1[i])*Srate);
       return ((son+sof)/2);
       }
   }
   else
   {
   f  =  (((flag[i]  &  4)/4)|((flag[i]  &  8)/8))+  (((flag[i]  &  16)/8)|((flag[i]  &
32)/16));
   switch (f)
   {
       case 1:
       if (SP>0)
           {
```

```
          ton = (long)(((Srate*(Dist+LL))/SP)+0.5);
          if (abs(TOF1[i]-ton)<DTe)
          {
                  sof = (((Dist+LL)/TOF1[i])*Srate);
                  son = (((Dist+LL)/ton)*Srate);
                  return ((son+sof)/2);
          }
          else
          {
                  return (((Dist+LL)/TOF1[i])*Srate);
          }
          }
          else
          {
          return (((Dist+LL)/TOF1[i])*Srate);
          }
      break;
      case 2:
      if (SP>0)
          {
          ton = (long)(((Srate*(Dist+LL))/SP)+0.5);
          if (abs(TON1[i]-ton)<DTe)
          {
                  son = (((Dist+LL)/TON1[i])*Srate);
                  sof = (((Dist+LL)/ton)*Srate);
                  return ((son+sof)/2);
          }
          else
          {
                  return (((Dist+LL)/TON1[i])*Srate);
          }
          }
          else
          {
          return (((Dist+LL)/TON1[i])*Srate);
          }
          break;
      case 3:
      if (SP>0)
      {
      return (SP);
      }
      else
      {
      return 0;
      }
      break;
  }
  }
  }
```

Figure 43. Procedure *speed()*

3.2.2.7. Procedure length()

The procedure *length()*, depicted in Figure 44, estimates the length of the vehicle detected by the inductive dual-loop detectors. The procedure length() was developed specifically for the needs of this project and is based on the data captured from two test locations in Portland, Oregon. In this procedure, the acceleration of the vehicles as they travels past the two loops is estimated. If the acceleration is more than an experimental threshold, a length calculation method other than averaging is employed.

```
   float length(int i, float s)
   {
   int f;
   float ls,lm;
   float ss;

   f = (((flag[i] & 128)/128)|((flag[i] & 256)/256))+(((flag[i] & 512)/256)|((flag[i]
& 1024)/512));
   ss = s/Srate;
   switch (f)
   {
       case 0:
       ls = (float)((TS1[i]*ss)-LL);
       lm = (float)((TM1[i]*ss)-LL);
       return ((ls+lm)/2);
       break;
       case 1:
       return (float)((TM1[i]*ss)-LL);
       break;
       case 2:
       return (float)((TS1[i]*ss)-LL);
       break;
       case 3:
       ls = (float)((TS1[i]*ss)-LL);
           lm = (float)((TM1[i]*ss)-LL);
           return ((ls+lm)/2);
       break;
   }
   }
   void Cflag(int i,float s, float l)
   {
   if (s < Smin) { flag[i] = flag[i]+4096;}
   if (s > Smax) { flag[i] = flag[i]+8192;}
   if (l < Lmin) { flag[i] = flag[i]+16384;}
   if (l > Lmax) { flag[i] = flag[i]+32768;}
   }
```

Figure 44. Procedure *length()*

3.2.2.8. Procedure Eflag()

The *Eflag()* procedure, depicted in Figure 45, calculates the error flags used by the speed calculation procedure and also written in the output data file. The thresholds for each error type are defined at the beginning of the program.

```
void Eflag(int i)
{
//sets error flag
flag[i] = 0;
if (TON1[i] < Temin) { flag[i] = flag[i]+4;}
if (TON1[i] > Temax) { flag[i] = flag[i]+8;}
if (TOF1[i] < Temin) { flag[i] = flag[i]+16;}
if (TOF1[i] > Temax) { flag[i] = flag[i]+32;}
if ((fabs(TOF1[i]-TON1[i])/TON1[i])> DTe) { flag[i] = flag[i]+64;}
if (TS1[i] < Tonmin) { flag[i] = flag[i]+128;}
if (TS1[i] > Tonmax) { flag[i] = flag[i]+256;}
if (TM1[i] < Tonmin) { flag[i] = flag[i]+512;}
if (TM1[i] > Tonmax) { flag[i] = flag[i]+102;}
if ((fabs(TM1[i]-TS1[i])/TS1[i])> DTo) { flag[i] = flag[i]+2048;}
if ((TON1[i] == 0) || (TOF1[i] == 0)) { flag[i] = flag[i]+262144;}

}
```

Figure 45. Procedure *Eflag()*

3.2.2.9. Procedure Cflag()

The *Cflag()* procedure, depicted in Figure 46, calculates some additional error flags that depend on calculated speed. This procedure has to be called after the *speed()* procedure since it requires vehicle speed as input.

```
void Cflag(int i,float s, float l)
{
if (s < Smin) { flag[i] = flag[i]+4096;}
if (s > Smax) { flag[i] = flag[i]+8192;}
if (l < Lmin) { flag[i] = flag[i]+16384;}
if (l > Lmax) { flag[i] = flag[i]+32768;}
}
```

Figure 46. Procedure *length()*

4.0 TRUCK COUNT UNIT ACCURACY ASSESSMENT

This chapter documents the performance testing of the TCU to validate its accuracy in estimating vehicle speed and length. The TCU platform developed was tested at two different locations on Portland, Oregon.

4.1. PROCEDURES AND ANALYSIS METHOD

The accuracy of the speeds and lengths generated by the TCU were assessed by comparison to the corresponding speeds and lengths obtained from high-speed video analysis. These videos were recorded at the two locations where dual inductive loops monitored by TCUs were installed in the city of Portland:

- I-205 northbound lanes near Stafford Road (Location 1),
- All lanes on 82nd Avenue approximately 0.1 miles south of the 82nd Avenue and Freemont intersection (Location 2).

The TCUs were installed in control cabinets near the dual loop installations at:

- The on-ramp to I-205 northbound from Stafford Road, exit
- 82nd Avenue and Freemont.

During the visits to all locations, the length of the loops and the distance between loops were measured. To help with these measurements, chalk lines perpendicular to the road were drawn on the road shoulder that were aligned with the beginning and end of each loop. Distances were then measured from these chalk lines. Stakes were used to mark the beginning of the first loop and end of the second loop to assist in the video analysis. Figure 47 shows the chalk lines and the stakes used for measurement and video analysis at location 1.

Figure 47. I-205 Northbound Dual Inductive Loop Markings

At all locations, the video camera was placed at an imaginary line perpendicular to the road and exactly in the middle of the two stakes. The high-speed video camera was able to record 240 frames per seconds for a maximum duration of 13 minutes. The high number of frames per second allowed accurate estimation of vehicle speeds.

By counting the number of video frames required to record a vehicle moving between the two stakes, vehicle speed could be estimated from the video using the following formula:

$$T_s = \textit{Vehicle travel time between the two stakes} = \frac{Number\ of\ frames}{240}\ (Seconds) \qquad (4.1)$$

$$\textit{Estimated Speed}\ (\frac{Distance}{Second}) = \frac{Distance\ between\ the\ two\ stakes}{T_s} \qquad (4.2)$$

Knowing the distance between the two stakes, vehicle lengths could be estimated from the video using the following formula:

$$\textit{Estimated Length} = \frac{Length\ on\ the\ video\ monitor\ \times Distance\ between\ the\ two\ stakes}{Distance\ between\ the\ two\ stakes\ on\ the\ video\ monitor} \qquad (4.3)$$

After matching the TCU speed and length estimates with the corresponding vehicles in the video, the percentage differences between the TCU speed and length estimates and the video analysis speed and length estimates for each vehicle were calculated.

The loop detector sensitivity for detecting objects is adjustable on an integer scale from one to seven, with a higher value resulting in higher sensitivity. The default sensitivity level for all three locations was four. The effect of sensitivity level on the length and speed estimates was tested at location 1. Sensitivity levels of three, four, and five were tested.

After estimating vehicle lengths, the vehicles could be classified into different vehicle classes. ODOT uses the following length-based classes:

- Passenger vehicles (PV) - less than 20 feet.
- Single unit trucks (SU) - between 20 and 35 feet.
- Combination trucks (CU) - between 36 and 60 feet.
- Multi-trailer trucks (MU) - greater than 61feet.

Corresponding vehicles from the video analysis and TCU data were classified and the counts in the various classes were compared.

4.2. LOCATION 1 RESULTS

Table 7 and Table 8 below show the average differences between the speed and length estimates from the video analysis, and the TCU speed and length estimates for each sensitivity level.

Table 7. Average differences at location 1 between the video analysis and TCU length estimates

Sensitivity Level	Number of Vehicles in Video	Number of Vehicles Analyzed	Avg. Length Difference (ft)
3	517	93	-0.69
4	513	30	-0.74
5	484	91	5.22

Table 8. Average differences at location 1 between the video analysis and TCU speed estimates

Sensitivity Level	Number of Vehicles in Video	Number of Vehicles Analyzed	Avg. Length Difference (ft)
3	517	93	-0.038
4	513	30	-1.164
5	484	91	-2.41

Figure 48, Figure 49 and Figure 50 depict histograms of length errors for each loop detector sensitivity level. Figure 51, Figure 52 and Figure 53 depict histograms of speed errors for each loop detector sensitivity level.

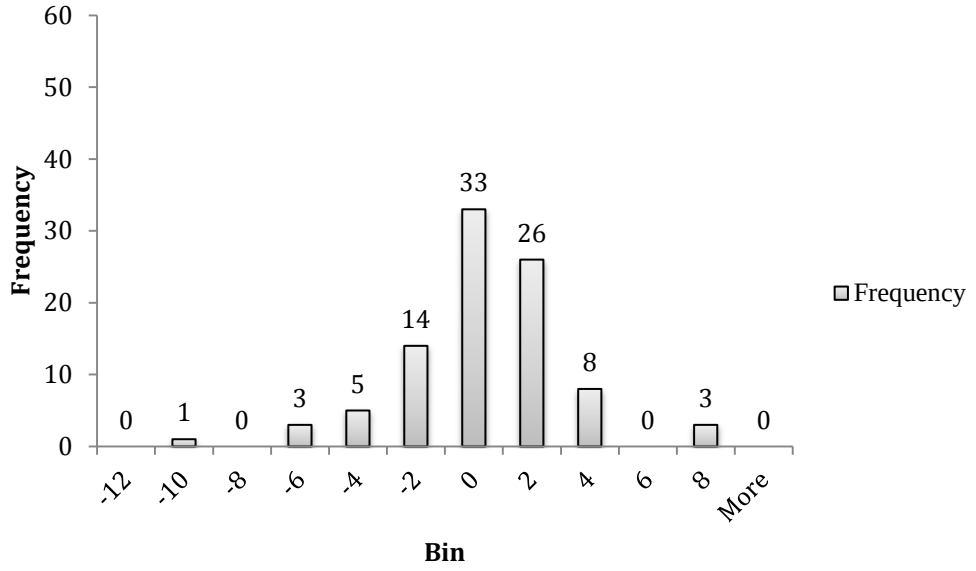

Figure 48. Histogram of Length Errors (in Feet) – Location 1, Sensitivity 3

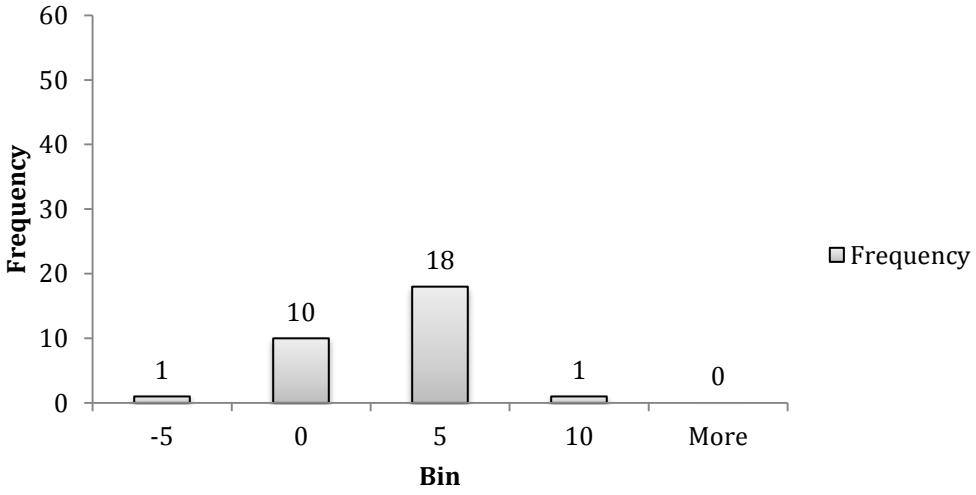

Figure 49. Histogram of Length Errors (in Feet) – Location 1, Sensitivity 4

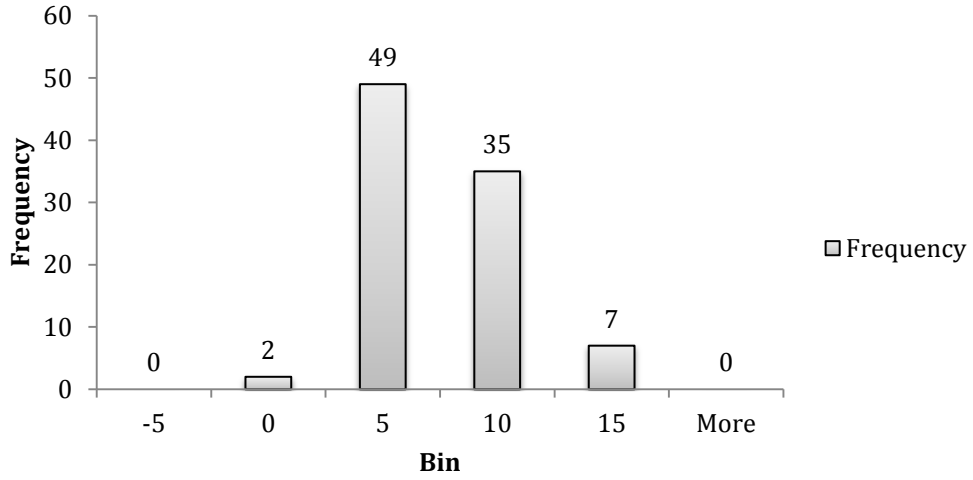

Figure 50. Histogram of Length Errors (in Feet) – Location 1, Sensitivity 5

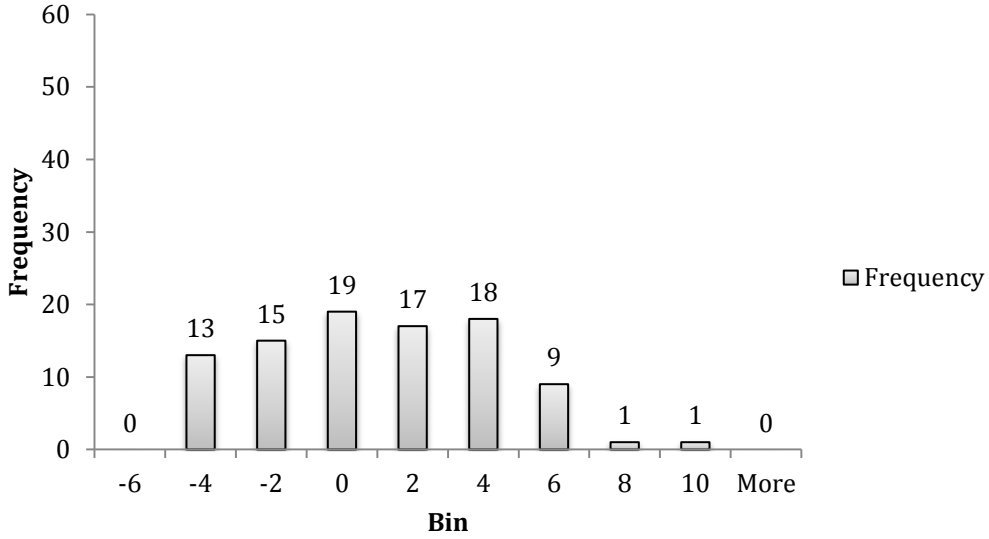

Figure 51. Histogram of Speed Errors (in Miles per Hour) – Location 1, Sensitivity 3

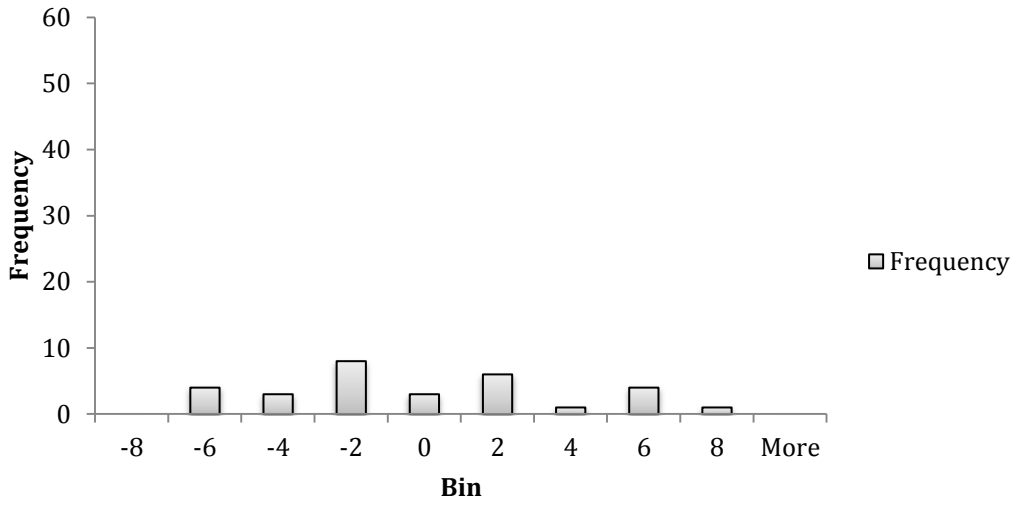

Figure 52. Histogram of Speed Errors (in Miles per Hour) – Location 1, Sensitivity 4

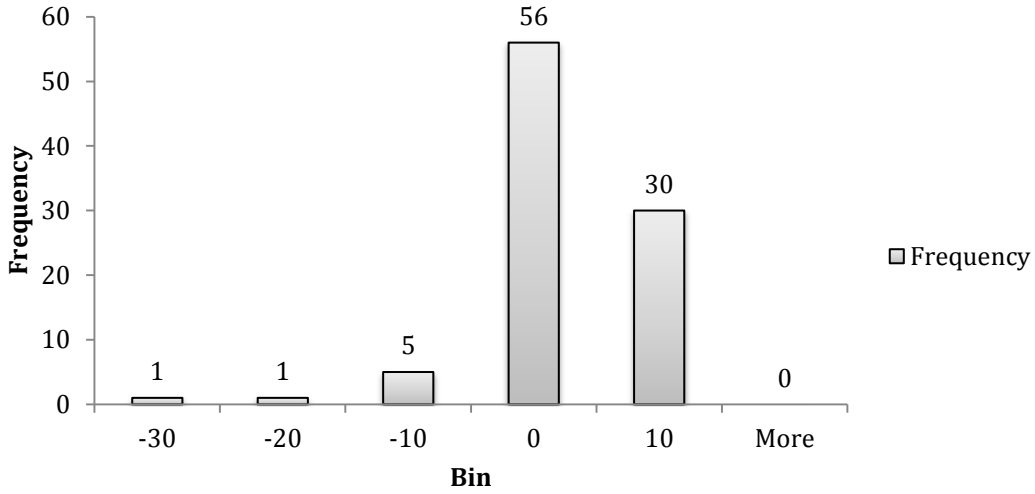

Figure 53. Histogram of Speed Errors (in Miles per Hour) – Location 1, Sensitivity 5

Figure 54, Figure 55 and Figure 56 depict a comparison between the vehicle classification results obtained through video analysis and TCU data for each loop detector sensitivity level.

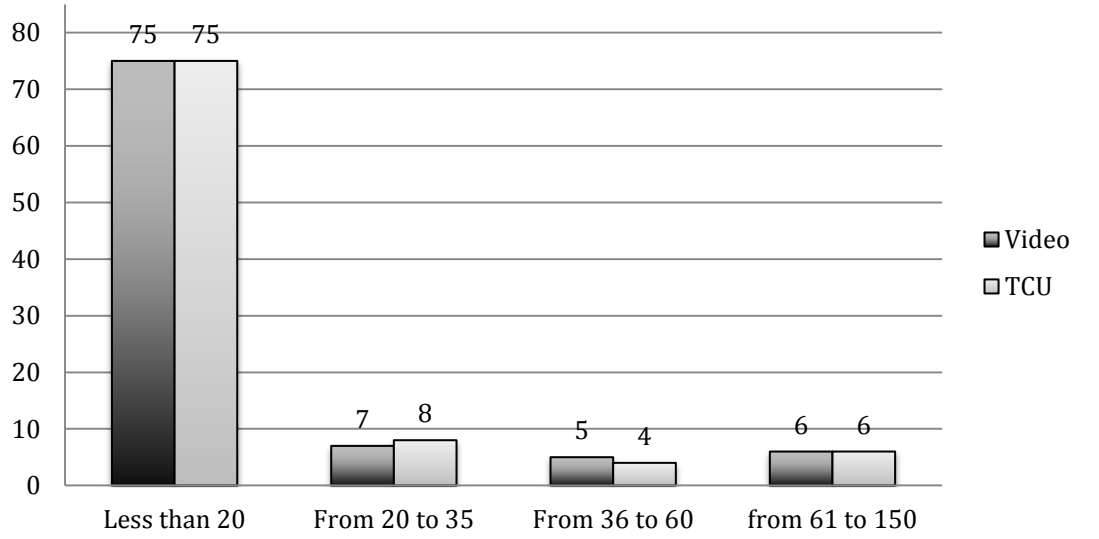

Figure 54. Comparison of Length Classification Between Video and TCU data – Location 1, Sensitivity 3

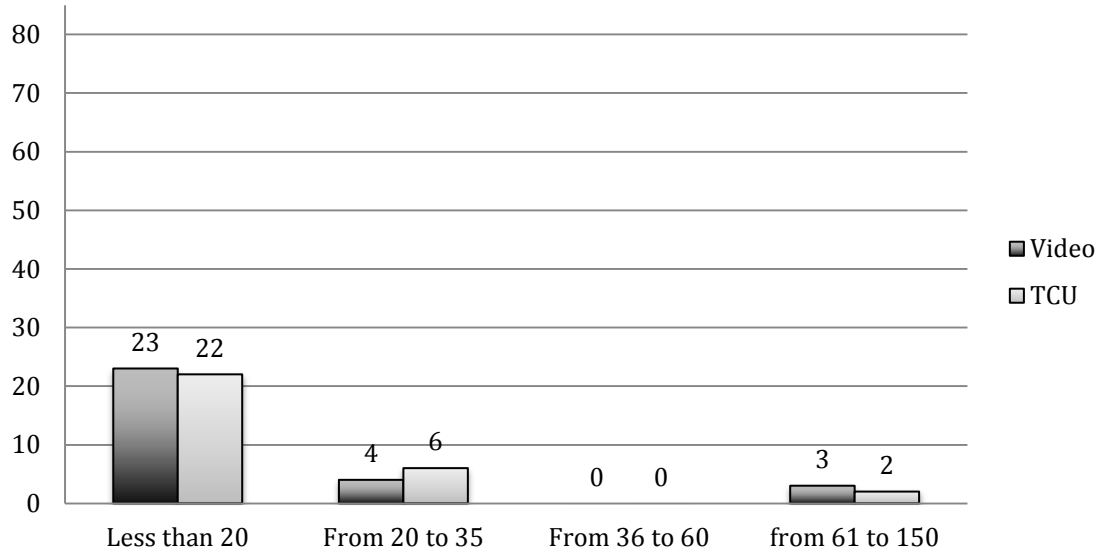

Figure 55. Comparison of Length Classification Between Video and TCU data – Location 1, Sensitivity 4

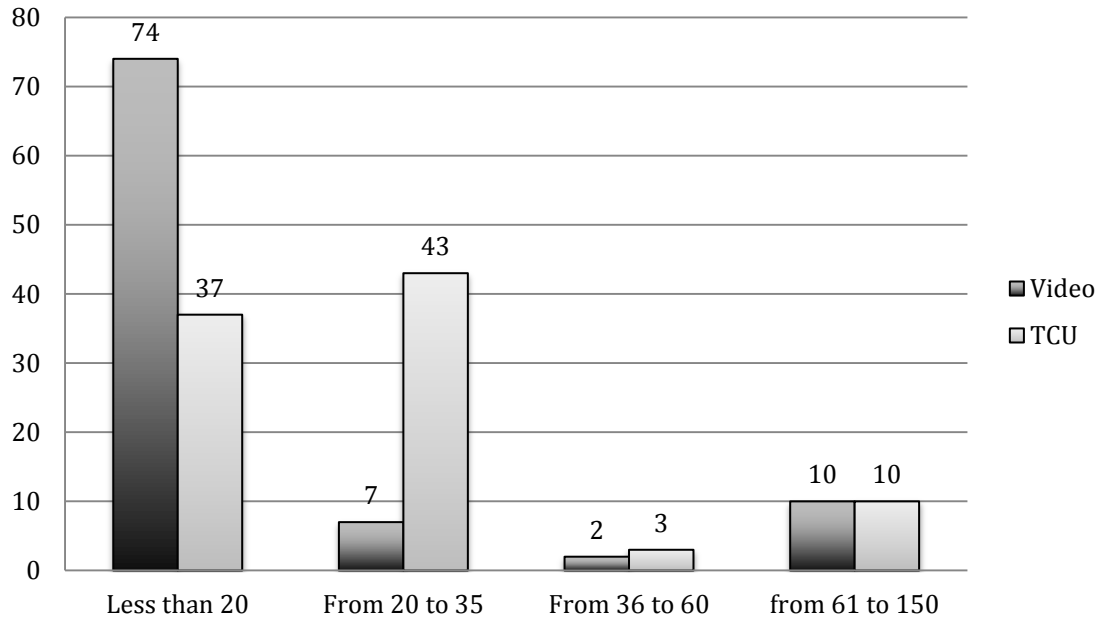

Figure 56. Comparison of Length Classification Between Video and TCU data – Location 1, Sensitivity 5

These results show that both sensitivity levels three and four produce consistent results within each bin level. Sensitivity five is clearly producing great differences and unacceptable vehicle classification results (assuming the video analysis as the *ground truth*). The default ODOT loop detector sensitivity level is four and the results support that at this sensitivity level the TCU performs accurately with respect to estimating vehicle length, speed, and classification.

4.3. LOCATION 2 RESULTS

Table 9 shows the average error in the calculation of speed and length for location 2. Figure 57 depicts the distribution of length according to ODOT's classification scheme. This analysis also confirmed that the TCU is able to estimate vehicle lengths and speeds with acceptable error.

Table 9. Summary of length and speed errors at location 2

Number of Vehicles in Video	Number of Vehicles Analyzed	Avg. Length Difference (ft)	Avg. Speed Error (mph)
885	37	2.94	3.28

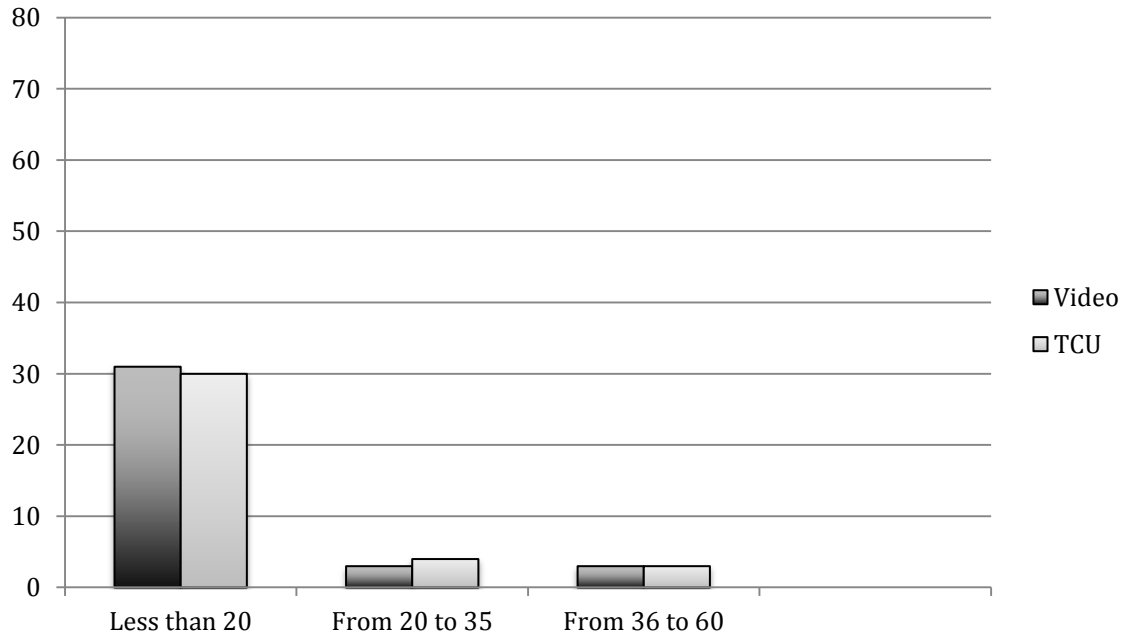

Figure 57. Comparison of Length Classification Between Video and TCU data – Location 2

Figure 48, Figure 49 and Figure 50 depict histograms of length errors for each loop detector sensitivity level. Figure 51, Figure 52 and Figure 53 depict histograms of speed errors for each loop detector sensitivity level.

Figure 58 depicts a histogram of length errors at location 2, whereas Figure 59 depicts a histograms of speed errors at location 2.

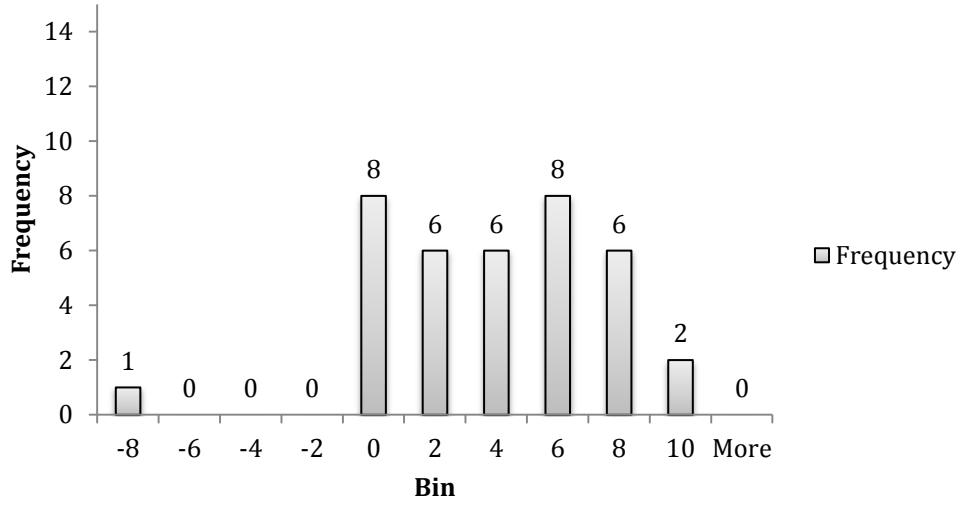

Figure 58. Histogram of Length Errors (in Feet) – Location 2

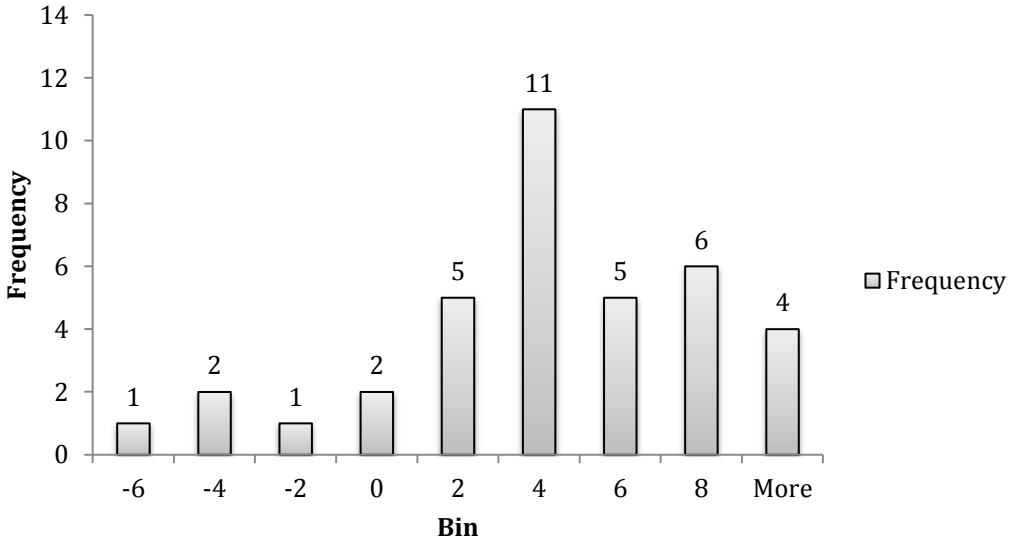

Figure 59. Histogram of Speed Errors (in Miles per Hour) – Location 2

5.0 CONCLUSIONS AND RECOMMENDATIONS

To be completed

6.0 REFERENCES

Acrylonitrile butadiene styrene. (2013, January 27). In *Wikipedia, the free encyclopedia*. Retrieved from http://en.wikipedia.org/w/index.php?title=Acrylonitrile_butadiene_styrene&oldid=534476303

Itekyala, S. R. (2010). *Vehicle Classification under Congestion using Dual Loop data*. University of Cincinnati.

Nihan, N. L., Wang, Y., & Cheevarunothai, P. (n.d.). Improving Dual-Loop Truck (and Speed) Data: Quick Detection of Malfunctioning Loops and Calculation of Required Adjustments.

Nihan, N., Wang, Y., & Zhang, X. (2005). *Improved Dual-Loop Detection System for Collecting Real-Time Truck Data*. Department of Civil Engineering Of University of Washington.

Nihan, Nancy L., Wang, Y., Cheevarunothai, P., & Northwest, T. (2006). *Improving dual-loop truck (and speed) data: quick detection of malfunctioning loops and calculation of required adjustments*. Citeseer. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.112.7244&rep=rep1&type=pdf

APPENDIX A