Title
Freeway Service Patrol (fsp) 1.1: The Analysis Software For The Fsp Project

Author
Petty, Karl

Publication Date
1995

CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

Freeway Service Patrol (FSP) 1.1: The Analysis Software for the FSP Project

Karl Petty

California PATH Research Report
UCB-ITS-PRR-95-20

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department of Transportation, Federal Highway Administration.

Report for MOU 91

June 1995

ISSN 1055-1425

FSP 1.1

The Analysis Software for the FSP Project

Karl Petty

FSP Project
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94720

Mailing for bug reports: `pettyk@eecs.berkeley.edu`

This manual is for the FSP and XFSP programs
June 4, 1995

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This manual is the reference manual for the **fsp** and **xfsp** programs. The **fsp** program is a software tool used to interrogate the data that was collected during the Freeway Service Patrol Evaluation Project. This program will perform diagnostics on the data, generate error reports, and make plots of various pieces of data. The program takes as it's input arguments a file that we shall call a runfile, an incident filter file, and an incident run number. The runfile contains all of the commands that the **fsp** program needs to run. The incident filter tells the program which incidents to filter out of the incident database and the incident run number is just an index for the output files. The **xfsp** program is a graphical user interface to the **fsp** program that was written in Tcl/Tk[1] and Expect[2]. This program allows the user to generate the runfile and the incident filter by clicking on various buttons and widgets with the mouse. Since the **xfsp** program is just a graphical user interface to the **fsp** program this manual will concentrate on explaining the different types of analysis that the **fsp** program will perform. The **xfsp** program is described in more detail in Chapter 3.

The **fsp** program generates quite a few different types of output. If you were to run the program on a complete data set the program would take about 12 hours and could possibly generate up to 8000 files. As a result, a large portion of the manual is going to be devoted to the interpretation of the various output files. A summary of the various types of output files and plots is given below:

- Loop Data:
  - Speed vs. Time plots
  - Counts vs. Time plots
  - Occupancy vs. Time plots
  - Delay (v-hr) vs. Time plots
  - Delay tables
  - Text reports of the data
  - Error reports on the data
  - Reports of dropout times
- Car Data:

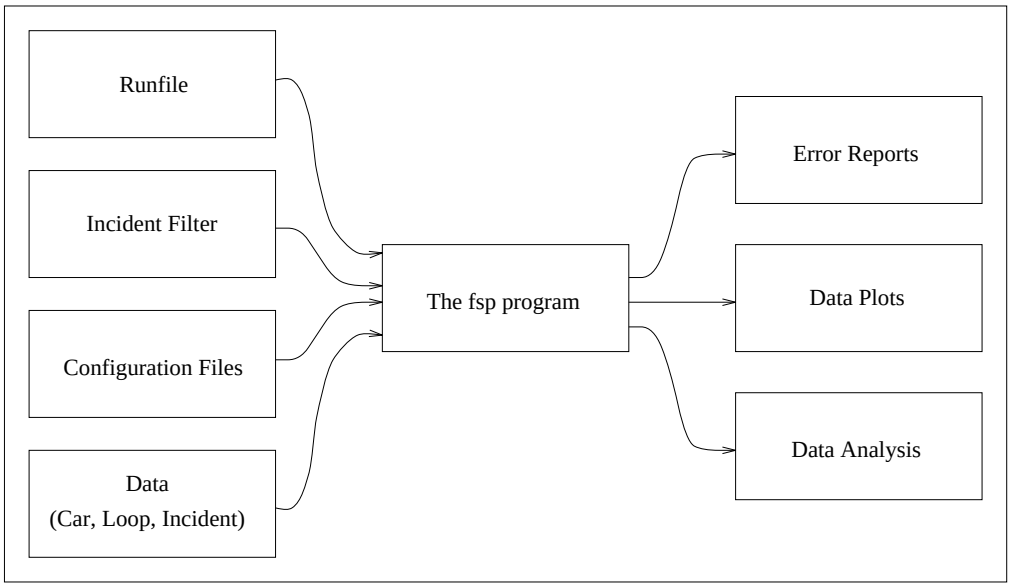

Figure 1.1: Basic Program Structure.

  - Latitude vs. Longitude plots of car trajectory
  - Speed vs. Time plots of car trajectory
  - Distance vs. Time plots of car trajectory
  - Speed vs. Distance plots of car trajectory
  - Link Travel Time vs. Starting Time
  - Plots of GPS data
  - Driver evaluations

- Incident Data:
  - Histograms of incident duration
  - Cumulative distribution of incident duration

- Data Analysis:
  - Delay per incident
  - Plot of delay per incident duration
  - Plots of correlation between car and incident data
  - Contour plots of delay on the freeway

Although this may seem too much to have to deal with, hopefully this manual will make everything seem clear.

Basically the whole system looks like Figure 1.1. The program takes as input a runfile, an incident filter file, various configuration files, and some data. It generates as output

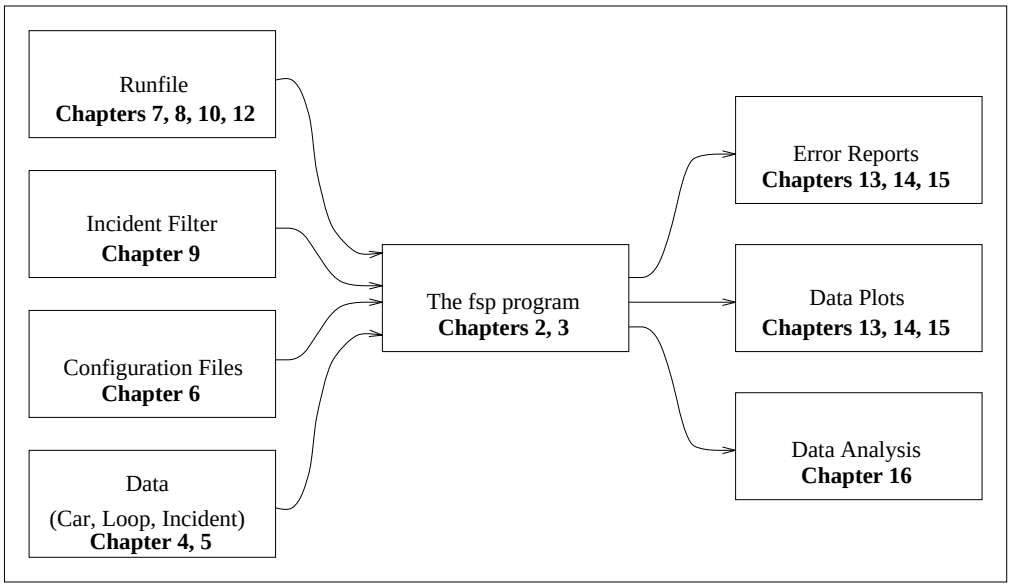

Figure 1.2: Basic Program Structure with Chapters.

various error reports, graphs, and tables. This manual is an attempt to describe in detail all of the various boxes listed in Figure 1.1.

The chapters in this manual almost follow the boxes in Figure 1.1. Chapter 2 deals with how to setup the **fsp** program and Chapter 3 deals with how to setup the **xfsp** program. Chapter 4 talks about the data that we collect during the course of the experiment and Chapter 5 talks about the problems associated with the data. Chapter 6 describes how the **fsp** program expects the data to be stored on the system. Chapters 7, 8, 10, and 11 discuss the different parameters that can be set in the runfile and Chapter 9 explains how to generate incident filters. In terms of program output, Chapter 13 explains the basics of how to view the output and where you can expect to find it. Chapters 14, 15 explain the various types of output that are generated from the loop and car data sets. Chapter 16 explains the output that is generated from the cross data analysis. Finally, Chapter 17 talks about how the **fsp** program fits into a larger picture. With this in mind I would like to relabel my diagram of the basic program structure to include the chapter numbers in the appropriate boxes. I have done this in Figure 1.2.

During the course of the Freeway Service Patrol there turned out to be a need to have more detailed data collection from the cars. What this basically means is that the format of the data from the cars changed in the middle of the experiment. I have pointed out in the manual where this can cause problems.

Although I wrote the **fsp** program, I never could have done it without the help of quite a few people. Probably the most important is Leon Chen. He wrote the code that reads in the binary loop data and converts it to an understandable format. The routines that process the loop data still use his code as a foundation. Kumud Sanwal wrote the routines to do the consistency fix on the loop data and part of this documentation was written by him as well. Hisham Noeimi and Dan Rydzewski came up with the format of the incident database which plays a big part in **fsp** program. They also had the thankless job of collecting all of the data.

Dr. Alex Skabardonis managed the project and wrote the final report. I would also like to acknowledge my advisor Professor Pravin Varaiya for his continuing support and many helpful suggestions. If you are looking for a summary of the results of the Freeway Service Patrol Evaluation Project then you should consult [3].

Please note that this software is currently in flux. Hence, the manual is not quite finished yet. There are bound to be typos, bad grammar, and even incorrect instructions. If you find that something doesn't work then please send me email at pettyk@eclair.eecs.berkeley.edu. I'm not guaranteeing that I'll be respond - it's just that it would be nice to know if there were any bugs.

Chapter 2

Setting Up And Running The fsp Program

The **fsp** program is rather large and complex. There are quite a few input files that need to be placed in the right spots, and quite a few input directories that need to be created. I have outlined the steps below to setup the **fsp** program, create the directories and finally to compile and execute the software. The installation procedure for the **xfsp** program is quite a bit harder than for the **fsp** program. As a result, all of Chapter 3 is devoted to installing and running the **xfsp** program.

The steps to generating the **fsp** code and running it are pretty straight forward:

1. Download the software.
2. Compile the program using the included makefile.
3. Install the configuration files using the included makefile.
4. Install some data.
5. Make a runfile.
6. Make an incident filter.
7. Run it.

In this chapter we will talk about the first 3 items in this list. The fourth item, installing some data, is discussed in Chapter 4, and the fifth and sixth items, making a runfile and an incident filter, are discussed in Chapters 7, 9, and 10.

2.1 Setting Up The fsp Program

Downloading the software is pretty straight forward. You simply need to download one file from the the software server at Berkeley. This is done via a mechanism called anonymous ftp. To use this simply type the following command:

```
ftp www-path.eecs.berkeley.edu
```

When the machine prompts you for a login name type in the word "anonymous." When you are prompted for a password type in your email address like: "me@some.machine.somewhere." This will let you in. I would suggested that you download any files named `README` and look at those first. The **fsp** software package is located under `/pub/PATH/FSP/Packages/fsp.1.1.tar.Z`. You should download this by first changing to that directory and then typing the command:

```
get fsp.1.1.tar.Z
```

This will download the file from the machine at Berkeley to your local machine. The way that the **fsp** program is distributed is in a compressed tar file. The file, on your machine, should look something like this:

```
clair 1: ls
fsp.1.1.tar.Z
```

To unpack the data simply type the following command:

```
clair 2: uncompress fsp.1.1.tar.Z
```

This will create a file called **fsp.1.1.tar** which you then need to "untar" with the following command:

```
clair 3: tar xvf fsp.1.1.tar
```

This last command will create a directory on your system named **fsp**. This directory will be referred to as the main **fsp** directory. Note that if you already have a directory named **fsp** then it might be overwritten by the tar command. A listing of the directory should look something like this:

```
clair 4: cd fsp
clair 5: ls
Makefile        Set1            fsp_src         xfsp_src
README.DOC      Set2            manual          xfspview_src
```

This directory has the following set of subdirectories:

1. Directory of source files for the **fsp** program (**fsp_src**).
2. Directory of source files for the **xfsp** program (**xfsp_src**).
3. Directory of source files for the **xfspview** program (**xfspview_src**).
4. Directory of configuration files for the before data set (**Set1**).
5. Directory of configuration files for the after data set (**Set2**).
6. Directory holding this manual (**manual**).

The directory will also include a makefile named **Makefile** and a documentation file named **README.DOC**. In order to compile the fsp program you need to do the follow steps:

1. Figure out where you want to place the data. This will be referred to by its makefile name, `FSP_DATA_DIR` (for the fsp data directory).

2. Manually create this directory.

3. Edit the makefile and follow the instructions in there. You will need to set the value of `FSP_DATA_DIR` to be the name of the directory that you just created.

4. Compile the **fsp** program by typing in the main directory:

```
clair 6: make fsp
```

   This will make the **fsp** program. Note that you don't have to change into the **fsp_src** directory for this to work - this should be done from the main **fsp** directory. You should see something like the following output:

```
clair 6: make fsp
cc -g  -c fsp.c
cc -g  -c compassc.c
cc -g  -c cparsec.c
cc -g  -c fsp_util.c
cc -g  -c makeprnc.c
cc -g  -c congpsc.c
cc -g  -c log_170c.c
cc -g  -c log_stat.c
cc -g  -c log_fsp.c
cc -g  -c log_util.c
cc -g  -c log_flow_plot.c
cc -g  -c inradc.c
cc -g  -c fsp_calc.c
cc -g  -c loop_util.c
cc -g  -c inc_util.c
cc -g  -c inc_pos.c
cc -g  -c inc_print.c
cc -g  -c fsp_neural.c
cc -g  -c fsp_mkavg.c
cc -g  -c fsp_dir.c
cc -o fsp -g fsp.o compassc.o cparsec.o fsp_util.o makeprnc.o
congpsc.o log_170c.o log_stat.o log_fsp.o log_util.o log_flow_plot.o
inradc.o fsp_calc.o loop_util.o inc_util.o inc_pos.o inc_print.o
fsp_neural.o fsp_mkavg.o fsp_dir.o -lm
chmod ugo+rx fsp
```

If this is not what happens and the **fsp** executable program has not been created then make sure that all of the libraries are in the appropriate place, that all the include files are around, and that the source files are there as well. If you still can't get it to compile then track down a hacker and ask them.

5. To install the configuration files type:

```
clair 7: make set1
or
clair 7: make set2
```

The first thing that this will do is to create all of the subdirectories that you will need to store the data. For an explanation of the various directories and where the data should be placed see Chapter 4. This command will also copy all of the files from the various configuration directories into the appropriate spots under the `FSP_DATA_DIR`. Note that this will overwrite any configuration files that you already have in these directories. If you don't want to run the install portion of the make program then that is fine, but you'll have to manually copy the configuration files to their appropriate place yourself because the **fsp** program expects them to be there. For example, if the value of `FSP_DATA_DIR` is `/home/clair0/PATH/FSP/Temp/kp1` then the output of the "make set1" command should be something like this:

```
clair 8: make set1

Making main directories:

    Making directory /home/clair0/PATH/FSP/Temp/kp1/Loopdata
    Making directory /home/clair0/PATH/FSP/Temp/kp1/Cardata
    Making directory /home/clair0/PATH/FSP/Temp/kp1/Incidents
    Making directory /home/clair0/PATH/FSP/Temp/kp1/Runfiles

Installing configuration files:

    Installing files from loop_config
        Making loop data subdirs...
        Copying loop configuration files...

    Installing files from car_config
        Making car data subdirs...
        Copying car configuration files...

    Installing files from inc_config
        Copying incident configuration files...

    Installing files from runfile_config
```

```
Copying runfile example files...

Done
```

6. Next, you need to install the **fsp** software. Most people like to have all of the executable programs in a few locations on their system. These are usually `/bin` or `/usr/local/bin`. On step 9 in the makefile there is a variable named `DEST` that you can define as the destination for the **fsp** executable. Once you have set the variable `DEST` in the makefile, to install the **fsp** software you simply type:

   ```
   make install_fsp
   ```

   This will copy the program over from the source directory to the destination directory. If you don't want to install the **fsp** program in some common directory but instead wish to leave the program in the source directory then that is fine but you'll need to set your path such that you can find the **fsp** program.

7. Note that the installation procedure for the **xfsp** program is given in Chapter 3.

2.2 Running The Program

Once you have complied the program, installed the configuration files, and installed some data then you can start running the program. For an explanation of how to install some data see Chapter 4. There are a few things to note:

- You can run the program from anywhere on your system (as long as you have your path set correctly) and it should still put the data in the correct spot.
- If you halt the program, by typing control-c, you might see weird files lying around. If the program exits normally then these files are deleted, but if you halt it then it won't get a chance to delete them. You can just delete them yourself or you can wait until the program is run again and it will delete them when it exits normally. These files are named garbage.raw.
- You need to specify on the command line the name of the runfile, the name of the incident filter, and a run number. The run number is used to name, or index, some of the output files. If you want to know more about the runfile then refer to Chapter 7.
- If you want to run through the loop data then that takes the longest amount of time.
- To run the program simply type:

  ```
  clair 9: fsp my.runfile my.inc.filter 0
  ```

  (or whatever your runfile and incident filter are called).

- I usually like to run the program in the main data directory. On my system this would be the directory: `/home/clair0/PATH/FSP/Set2`.

Chapter 3

Setting Up And Running The xfsp Program

The **xfsp** program is a graphical user interface to the **fsp** program. It will allow you to create the runfile and the incident filter that the **fsp** program requires by pointing and clicking on various buttons and widgets with the mouse. It will also collect the output for you and display it in a window on the screen. There is an extensive help system within **xfsp** that should allow the user to control the **fsp** program without much need for this manual. This chapter probably should have been a little bit later in the manual because it assumes that you know something about the way the **fsp** program works. You can probably skip most parts of this chapter at first and then come back to them later. The part that you should probably read is the section on installing the **xfsp** program.

3.1 Introduction To The xfsp Program

The **xfsp** program does something really simple: it allows the user to generate a runfile and an incident filter for the **fsp** program. Once the runfile and the incident filter have been created, the **xfsp** program will execute the **fsp** program, collect the output and then display it on the screen. One can think of the **xfsp** program as being a "wrapper" for the **fsp** program: the **fsp** program is what does all of the processing work and the **xfsp** program is what deals with the user. One way to view this graphically is in Figure 3.1.

Figure 3.1 shows the **xfsp** program being in control of the **fsp** program: the **xfsp** program tells the **fsp** program what to do and then reads the output back from it. On the same figure is another program named **xfspview**. **Xfspview** is a program that allows the user to browse through the output of the **fsp** program by simply clicking on buttons. The **xfsp** program starts up the **xfspview** program every time that it runs the **fsp** program. The **xfsp** and **xfspview** programs are what the normal user will be using.

3.2 Support Software For The xfsp Program

The **xfsp** and **xfspview** programs were written on top of two different software tools: Tcl/Tk, that was written by John Ousterhout at the University of California at Berkeley[1], and Expect,

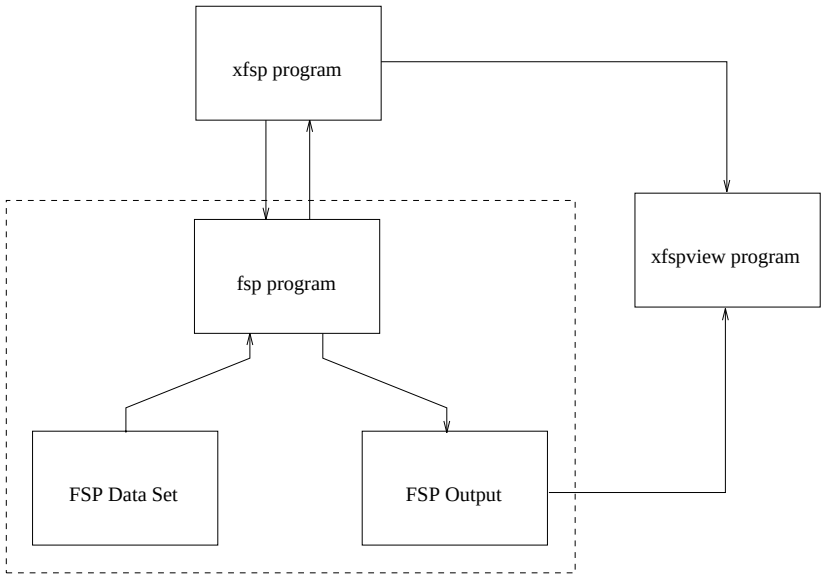

Figure 3.1: Relationship Between **fsp** and **xfsp** Programs.

that was written by Don Libes at the National Institute of Standards and Technology[2]. Tcl/Tk is a very powerful command language that lets you manipulate graphical objects like windows, buttons, sliderbars, etc. Expect is also a command language that is built on top of Tcl/Tk that allows the **xfsp** script to control the **fsp** program. In order to run the **xfsp** program you need to have Tcl/Tk and Expect on your system.

A list of the software packages that you need to download or already have installed is given below:

Tcl7.3 This is the "Tool Command Language" that is a simple script writing language for controlling and extending applications. It is one half of the Tcl/Tk package that is used to generate the graphical user interface (GUI) that the user will interact with.

Tk3.6 This is a toolkit for the X Window System. This allows you to manipulate various objects in X to create really nice graphical user interfaces. This is the second half of the Tcl/Tk package.

Expect 5 This is a program that allows scripts to interact with other programs. This will allow us to start up a program in the background and then collect it's output to a window. We will use this feature when the **xfsp** program starts up the **fsp** program. It is built on top of Tcl/Tk and should already be available on most systems in the form of the program called **expectk**.

xgraph and gnuplot (version 3.5) Xgraph and gnuplot are two standard pieces of software on any Unix system. Xgraph comes with the X Window System distribution which is now the windowing standard on workstations. Gnuplot is a plotting program that comes with GNU software. These two programs are used by the **xfsp** and **xfspview** programs to generate plots and graphs. The **fsp** program actually generates files that can be read

directly into gnuplot. Since these two pieces of software should be on your system already this chapter will not discuss installing them.

If you have access to the Internet then you can download these programs via anonymous ftp from many different sites around the country. The software packages are also located on the file server at UC Berkeley along with the **fsp** code so that you can download them easily. An important thing to note about all of these programs is that they are free and they are probably installed on your system already. If they aren't then there are steps below which you can follow to download and install these programs.

3.3 Installing The Support Software

Since there are a few different software packages, and the installation procedure can be a little confusing, the following discussion is provided to assist the user. This section will deal specifically with installing the *support software* for the **xfsp** program. Section 3.4 will deal with installing the **xfsp** program itself.

The most important thing that you can do at this point is to check and see if Tcl/Tk and/or Expect are already on your system. Specifically what we are looking for is the program called **expectk**. This is a version of Expect that was built with the Tcl/Tk support included. If this program is on your system them you can skip all the way to Section 3.4 and install the **xfsp** program. To find out if you have the **expectk** program on your system can type the following:

```
pettyk 1: which expectk
/usr/sww/bin/expectk
```

This tells me that the program **expectk** is located in `/usr/sww/bin/expectk`. If you type this command and it says something else about not being able to find the program then you should talk to your system administrator and see if it's been stored in an odd place.

If you don't have this program installed, then you need to download and install the programs yourself. An overview of the steps that you need to take to install the support software are as follows:

1. Download Tcl7.3, compile and install.

2. Download Tk3.6, compile and install.

3. Download Expect 5.16, compile and install.

Installing these packages isn't that hard. They were designed to be used by a whole variety of people and machine types so they are extremely easy to use. For the most part, you will only have to type three commands to completely compile and install each packages.

The final goal of installing the software packages is a program called **expectk**. This is a shell that the **xfsp** program will call to interpret it's commands.

3.3.1 Step 1: Creating The Directory Structure

The hardest part is figuring out where to place the packages in your directory structure. If you are installing the software as a system administrator then you will probably place the packages in `/usr/tools` or `/usr/local`. In that case you can probably figure out how to set up the links yourself so I won't tell you how to do this. The case that I will explain is when you aren't the system administrator. In that case, you will want to create a directory to hold all of the packages including the **fsp** package. It is very important that all of the packages reside in the same directory. The directory that I have chosen for this example is `/home/clair1/FSP`. Note that you will need approximately 32 megabytes just to hold the software packages.

You will also need to figure out where the executable files and libraries will reside. Once again, if you are the system administrator then there are a few obvious places for these files like `/usr/local/{bin,lib}`. But if you aren't the system administrator then I would recommend that you create the appropriate directories to hold these files in the current directory. In my case, the correct directory is `/home/clair1/FSP`. So in this directory I will create four subdirectories called: bin, lib, include and man with the following `mkdir` command:

```
clair 1: pwd
/home/clair1/FSP

clair 2: mkdir bin lib include man

clair 3: ls
bin     include lib     man
```

These will be the directories that will hold the executables, libraries and manual pages.

3.3.2 Step 2: Downloading The Software

Table 3.1 gives a list of where you can look to find the various software packages. Note that you can also download the software from the file server that contains the **fsp** and **xfsp** code which is currently `www-path.eecs.berkeley.edu`. To download software via anonymous ftp you only need to ftp to the site and then give your login name as "anonymous." When the system prompts you for a password simply type in your e-mail address.

Once the software has been downloaded to the desired directory your listing should look like the following:

```
clair 4: pwd
/home/clair1/FSP

clair 5: ls
bin             fsp.1.1.tar.Z   lib             tcl7.3.tar.Z
expect.tar.Z    include         man             tk3.6.tar.Z
```

At this point you should uncompress and untar all of the files. This is done by using the following commands:

Software package	Anonymous ftp site	Package path and name[1]
Tcl7.3	ftp.cs.berkeley.edu	/ucb/tcl/tcl7.3.tar.Z
	ftp.neosoft.com	/pub/tcl/distrib/tcl7.3.tar.gz
	ftp.uu.net	/languages/tcl/tcl7.3.tar.Z
	www-path.eecs.berkeley.edu	/pub/PATH/FSP/Packages/tcl7.3.tar.Z
Tk3.6	ftp.cs.berkeley.edu	/ucb/tcl/tk3.6.tar.Z
	ftp.neosoft.com	/pub/tcl/distrib/tk3.6.tar.gz
	ftp.uu.net	/languages/tcl/tk3.6.tar.Z
	www-path.eecs.berkeley.edu	/pub/PATH/FSP/Packages/tk3.6.tar.Z
Expect 5	ftp.cme.nist.gov	/pub/expect/expect.tar.Z
	www-path.eecs.berkeley.edu	/pub/PATH/FSP/Packages/expect.tar.Z
fsp	www-path.eecs.berkeley.edu	/pub/PATH/FSP/Packages/fsp.1.1.tar.Z
Study Area Map	www-path.eecs.berkeley.edu	/pub/PATH/FSP/Packages/freeway.ps

Table 3.1: Anonymous ftp sites for the software packages.

```
uncompress expect.tar.Z
tar xvf expect.tar
```

on each software package. After you have untar'ed the packages you can delete the tar files themselves because they are just wasting disk space.

3.3.3 Step 3: Installing Tcl 7.3

The steps for installing all of these packages is going to be just about the same:

1. cd into the package directory.
2. Read the documentation file (`README` or `INSTALL`).
3. Follow the instructions.

If you want to skip reading the documentation and just trust what I tell you then it should save you a lot of time. Everything that I tell you should work just fine, but if something goes wrong then you'll have to refer to their notes to correct the problem. The generic steps that you need to follow to install these, and quite a few other, packages are:

1. Run the **configure** program.
2. Run the **make** program.
3. Run the **make** program with the **install** option.

One nice feature that all of the software packages come with is a program called **configure**. This program will look at your system and figure out where all of your files are located and it will figure out what needs to be done to compile programs on your machine. This simplifies your job quite a bit. The only thing that you need to tell the **configure** program is where the executable and library directories (bin and lib) are located as well as the data and manual directories (include and man). In my directory structure, and hopefully in yours as well, these are both the same: `/home/clair1/FSP`. The complete command looks like this:

```
./configure --prefix=/home/clair1/FSP --exec_prefix=/home/clair1/FSP
```

Note that you need to put the `./` before the **configure** program so that you will be sure that you are running the local version and not the system version. Also note that you have to be in the Tcl directory to run this and the following commands. This will generate a lot of uninteresting output that you can just ignore. Once it is done you should attempt to compile the Tcl package by typing:

```
make
```

Finally, if this succeeds then you should install the package by typing:

```
make install
```

Once this step has completed you are done installing the Tcl package. If any of these steps don't succeed then you should examine the output and attempt to fix it yourself. If that fails then find somebody that knows about make files and ask them.

3.3.4 Step 4: Installing Tk 3.6

The installation for this package should be the same as for the Tcl package. Just cd into the tk3.6 directory and type the following commands:

```
./configure --prefix=/home/clair1/FSP --exec_prefix=/home/clair1/FSP
make
make install
```

These steps should generate a lot of output that you can just ignore. If the installation of the Tcl 7.3 package came off without a hitch then this package should be just as easy.

3.3.5 Step 5: Installing Expect 5

The installation for this package should be the same as for the Tcl package. Just cd into the expect-5.16 directory and type the following commands:

```
./configure --prefix=/home/clair1/FSP --exec_prefix=/home/clair1/FSP
make
make install
```

If the installation of the Tcl 7.3 package came off without a hitch then this package should be just as easy.

3.3.6 Step 6: Finishing Up

Once the software packages have been installed you can delete all of the intermediate files that were created by the various installation procedures. This is done by using the following command in each one of the package directories:

```
make clean
```

This will delete all of the files that you don't need.

In order to use the Tcl/Tk packages you need to set a few environment variables. These environment variables tell the Tcl/Tk programs where they can find the support programs that they need to run properly.

You will notice that when you installed Extended Tcl that it created two different directories in your main directory: These directories hold the support programs and files for the main Tcl/Tk program **wishx**. These are the directories that Tcl/Tk programs need to know about. We can set the appropriate environment variables with the following commands:

```
setenv TCL_LIBRARY /home/clair1/FSP/lib/tcl
setenv TK_LIBRARY /home/clair1/FSP/lib/tk
```

You should probably put these two statements in your `.cshrc` file so that these environment variables are set every time that you log on to your system. The Tcl/Tk programs will not run properly without them.

The last thing that needs to be done is you need to set the permissions on the files such that other people can use them. To do this you need to be in the main directory which on my system is `/home/clair1/FSP`. You need to run the following commands to open up the files for other people to use:

```
chmod -R ugo+rxX bin
chmod -R ugo+rX lib include man
```

This will enable different users to be able to execute the programs.

3.4 Installing The xfsp Software

Once you have the software packages installed, installing the **xfsp** software should be quite simple. Note that you absolutely have to have the above support packages installed in order for the **xfsp** program to work. The steps involved in installing the **xfsp** software are as follows:

1. Edit the makefile in the main fsp directory and follow the steps outlined there.
2. Assemble the program by typing **make xfsp**.
3. Install the program by typing **make install_xfsp**.
4. Set up an environment variable and correct your path.

The makefile in the main fsp directory is appropriately named **Makefile**. You should edit this file and simply read the steps listed. The important step for the **xfsp** installation is step 8. In this step you supply a complete path to the executable program **expectk** that you just compiled in Section 3.3. In the example above the complete path is `/home/clair1/FSP/bin/expectk`. So in the makefile for step 8 you would put the following line:

```
EXPECTK_EXE_PATH = /home/clair1/FSP/bin/expectk
```

The most important thing to note here is that the complete path to the **expectk** program, including the program name, has to be less than 32 characters. This is a quirk of Unix that is completely out of our control. In our example above the total path is 26 characters long and so we are fine. If the path is longer than 32 characters then you have to find a way to make it shorter. One way to do this is to create a link from one of the common executable directories to this file. The most obvious choices for the common executable directory are `/bin` or `/usr/local/bin`. In order to do this you must have root access.

Let's assume for a minute that the directory that we had placed our executables into had caused the complete path for the **wishx** program to have more than 32 characters. And let's also assume that we made a link from `/usr/local/bin/xfsp` to our executable file in `/home/clair1/FSP/bin/xfsp`. In that case, we would put the following line in the makefile:

```
EXPECTK_EXE_PATH = /usr/local/bin/xfsp
```

Note that this is the location of the link and not the actual file. Once we have completed the steps outlined in the makefile we can assemble the program by running the makefile on the **xfsp** program:

```
make xfsp
```

Note that this should be run in the main fsp directory, not the xfsp_src directory. Finally, we can install the **xfsp** and **xfspview** programs in the destination directory by running the makefile with the install option:

```
make install_xfsp
```

This should place the **xfsp** and **xfspview** programs in the executable directory.

Now that the **xfsp** program is in place you need to set an environment variable so that the program will know where the support files (help files, pictures, tables, etc.) are located. This is done by setting the environment variable `XFSP_DIRECTORY` to the main fsp directory. So in our example this would be done like this:

```
setenv XFSP_DIRECTORY /home/clair1/FSP/fsp.1.1
```

Just like the environment variables for the Tcl/Tk package, you should probably put this statement in your `.cshrc` file to make sure that it is set every time that you log on to your system. Finally, you have to set your path to include the **xfsp** program. If you installed the **xfsp** program in the same place that you installed the **expectk** program, like we did in this example, then you don't need to do anything else. If you didn't then you need to execute a statement like:

```
set path = ($path /home/clair1/FSP/bin)
```

Once again, something like this can be placed in your `.cshrc` file so that it is set every time you login.

3.5 Running The xfsp Software

Now that the software has all been installed you can run the **xfsp** program. You might want to first make sure that the following steps have been completed:

1. The environment variables `XFSP_DIRECTORY`, `TCL_LIBRARY` and `TK_LIBRARY` have all been set to their proper values.
2. Your path has been set to include the directories that hold the **expectk**, **xfsp**, **xfspview** and **fsp** programs.
3. You are currently in a directory that you have write access to. This is needed because the **xfsp** will create some files there.

If all of this has been done then you should be able to run the **xfsp** program by simply typing:

```
xfsp
```

If everything is set up properly then the **xfsp** control window will pop up onto the screen. It looks something like Figure 3.2.

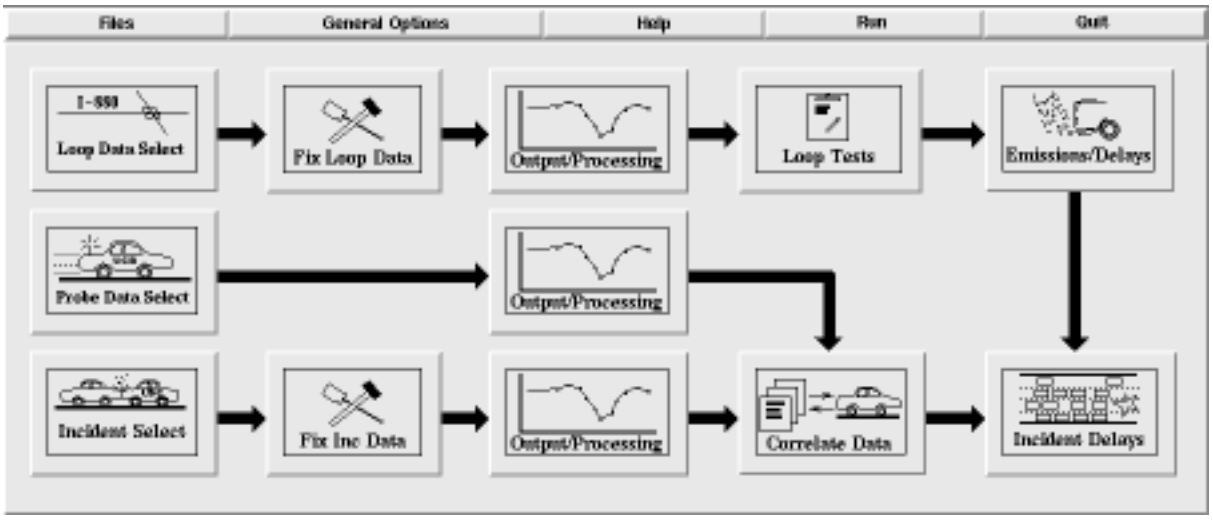

Figure 3.2: Main **xfsp** Window.

There are two distinct part of the **xfsp** control window. The top part consists of a menu bar with 5 buttons in it that control the various administrative parts of the program like file manipulation, the main help screens, exiting the program, etc. The large window right below the menu bar is the data flow window. Each of the buttons in the data flow window has

various options "underneath" it. If you click on one of the buttons then a window will pop up with the options specific to that button's function. For example, there is a button called "Fix Loop Data." If you click on that button then the window that holds the options that deal with fixing the loop data will pop up. Once you are done setting those options then you can pop that window down and play with some different options. For every button and every option there is either a help screen or an explanation button. You should probably just spend some time reading the explanations given for various options. The sections that follow will point out some of the details about that **xfsp** program that you should definitely be aware of.

3.5.1 The Layout Of The xfsp Window

In the **xfsp** data flow window there are three rows of buttons. These rows, from top to bottom, correspond to the various data sources that we have: the loop data, the probe vehicle data, and the incident database. The basic flow for each of the rows is the same:

- The buttons on the far left side of the data flow window allow you select the data that you would like to process.
- The buttons in the second column from the left allow you to apply various fixes to the data. For example, under the loop data fix button you can choose whether or not to fill in the holes in the loop data.
- The buttons in the middle column allow you to choose what kind of processing you would like to do on the various sets of data.
- In the fourth column you can choose to do various integrity tests on the loop data and/or indicate whether the program should attempt to correlate the incident database with the probe vehicle data.
- Finally, the last column of buttons deal with calculating the delay for each incident.

The layout of the data flow window was chosen to reflect the actual flow of data inside the **fsp** program. The arrows indicate the how the data flows (which is mostly from left to right). The hope is that this would help the user understand the function of each of the options.

3.5.2 Using The Fileselector

There are few times within the **xfsp** program that you will need to tell the program what file you are referring to. For example, you might want to use a setup file (Described in Section 3.5.3) to load some options. Or you might want to save the current runfile and incident filter without running the **fsp** program. In all cases, file selection is done though a piece of code called a file selector. The file selector window is shown in Figure 3.3.

The file selector reads in the contents of the current directory and displays them in the large window. The user is then allowed to select a file by either typing in a file name and then clicking on "OK" or by clicking twice really fast on one of the files in the list. Either way, the selected file is the passed back to the application which then uses it. You can traverse directories by either selecting ".." in the file list window or by typing a directory name in the

Figure 3.3: The **xfsp** File Selector Window.

entry window and hitting return. If you decide that you don't want to continue with whatever option brought up the fileselector window then you can simply click on the "Cancel" button to cancel everything.

3.5.3 Using Setup Files

The first thing that you will notice about the **xfsp** program is that there are quite a few options (approximately 120). The first thing that I noticed about using the program is that it's a complete pain to have to respecify all of the options that you desire every time that you start up the program (the program starts up with the parameters set to their default settings). Therefore, to easy this burden, the **xfsp** program can create and reload special files called setup files. A setup file is simply a file that the **xfsp** program creates that holds the current settings of all of the various options. So if you are doing a particular type of analysis and you have the parameters set a certain way then you might want to save these settings in a setup file so that you can run the program later and not have to respecify everything.

I would suggest that you utilize the setup files because they save quite a bit of time. What I do is I have one setup file for the before study data set and one for the after. Saving and loading the setup files can be done through the button names "Files" in the menu bar at the top of the control window.

3.5.4 Running The fsp Program

Once you have set up the options the way that you like you can have the **xfsp** program run the **fsp** program. This is done by choosing "Run" in the menu bar at the top of the **xfsp** control window. This will pop up a run window that looks like Figure 3.4 with a few options in it. Once you have set these then you can choose "Run" in the run window and the program will

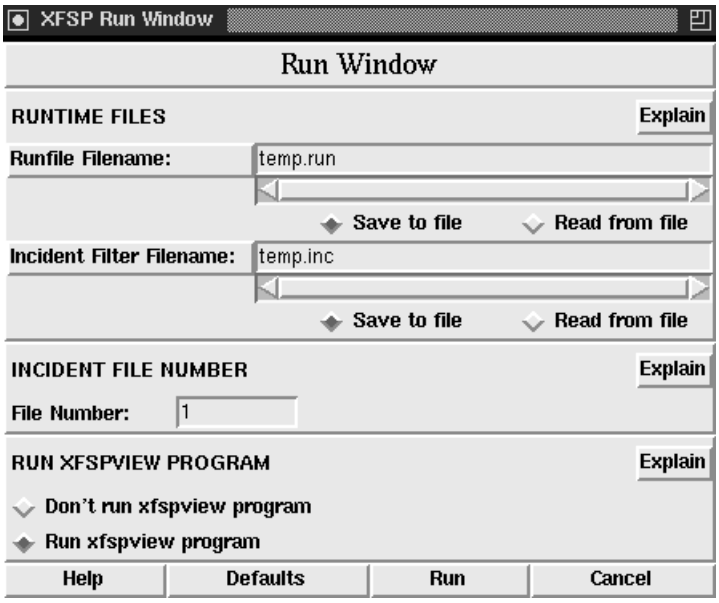

Figure 3.4: The **xfsp** Run Window.

start. When this happens the run window will pop down and an output window will pop up. An example of an output window is given in Figure 3.5.

The output window will collect the output from the **fsp** program and display it in the text area. At the bottom of the output window are a few buttons that allow you to save the output to a file or print it to a printer. The file that you can save to is called `fsp.out.X` and it is placed in the current directory (the "X" in the filename is the run number). The printer that you can print to is determined by the printer option in the **xfsp** program. These buttons should only be used once the **fsp** program has finished. You will know that the **fsp** program has finished processing normally because it spits out the following line:

```
******              END OF PROGRAM              ******
```

If the program does not finish successfully but encounters some sort of error, then it will say something bad and not spit out this line. When the **fsp** program gets done running (whether it finished successfully or not) the **xfspview** program will pop up (Section 3.5.5).

You should note that if you are doing any processing on the loop data or the car data that this might take a long time. In cases like this the **xfsp** program is not really very useful as an interactive tool. What you might want to do, instead of starting the **fsp** program from inside the **xfsp** program, is to save the runfile and incident filter that you have made and to then run the **fsp** program from the command line and put it in the background. You could even redirect the output to a file for later viewing. This would work really well when the program is expected to take 4 or 5 hours (which can happen).

3.5.5 Using The xfspview Program

The **xfspview** program is a graphical user interface to the data that the **fsp** program generates. The **xfspview** program can be viewed as a sister program to the **xfsp** program: the **xfspview**

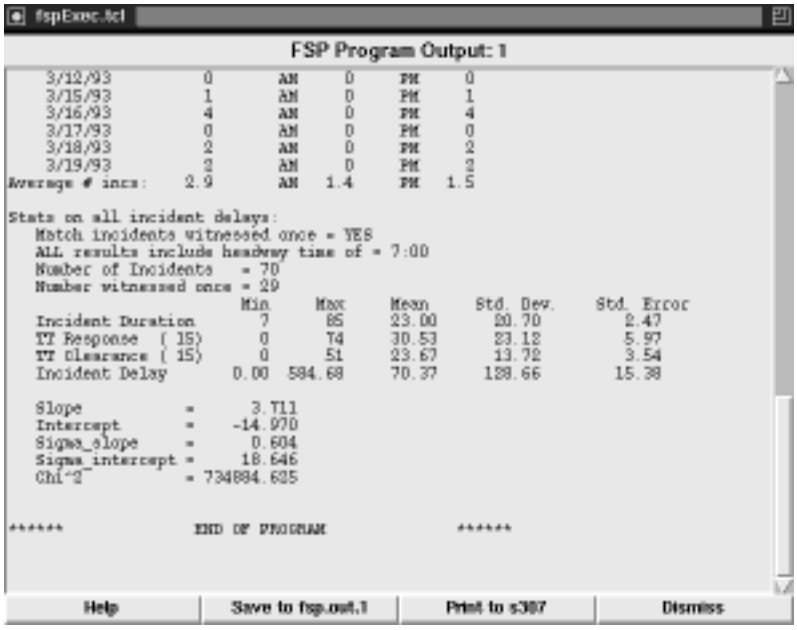

Figure 3.5: The **xfsp** Output Window.

program gives the user an easy way to view the output files that the **fsp** program has generated. The **xfspview** window is given in Figure 3.6.

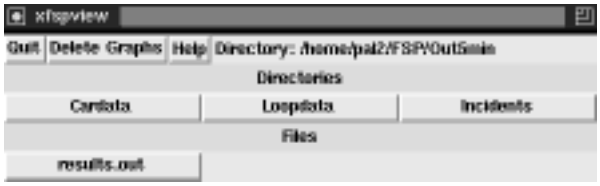

Figure 3.6: The **xfspview** Window.

The **xfspview** program is run in one of two ways:

- It can be run in stand alone mode where the user starts the **xfspview** program up from the command line. When the program is run this way the user needs to supply as the one argument the main output directory. This is done by simply typing the program name on the command line followed by the name of the output directory. Something like: `xfspview /home/data/Out5min`.
- It can be started up automatically by the **xfsp** program. After the **xfsp** program has run the **fsp** program it will start up the **xfspview** program if the user so requests. Whether this is done or not is set by a button in the "Run Window." The last panel of the "Run Window" allows the user to choose whether to run the **xfspview** program after the fsp program is done.

The **xfspview** has three main sections of buttons that are stacked vertically. The top section is a row of buttons labeled "Quit," "Delete Graphs," and "Help." These button

do fairly obvious tasks. The only potentially confusing button is the "Delete Graphs" button and we will talk about that a little bit later. The middle section of buttons has a title above it that says "Directories" and the bottom section of buttons has a title above it that says "Files." What the **xfspview** program does is it starts off in a directory and it reads all of the entries in the directory table. If an entry corresponds to a directory then a button is created with that directory name as it's label and it is placed in the middle section. If an entry is a file then a button is created for that file and it is placed in the bottom row. If the program starts out with a main output directory being passed to it (actually, it better start out with a main output directory or it will not work) then the middle section of buttons should read "Cardata," "Loopdata," and "Incidents." These are the main directories that the **fsp** program makes under the main output directory.

When you click on one of the buttons in the middle section (one of the buttons corresponding to a directory) then the program will change down into that directory and then reread all of the files. So if you were to choose "Loopdata" then the program will change into the loop data directory and it will read the directory table and create new buttons corresponding to the directories and the files. Note that whenever you are in a directory below the main directory that there will always be a button labeled `<Up>` that will take you to the directory directly above.

Whenever you click on one of the file buttons the program will either display the text file on the screen, generate a graph using the program **xgraph**, or it will generate a graph using the **gnuplot** program. The choice the program makes depends on what type of file you have selected. A more detailed description is given in the help windows of the **xfspview** program.

Chapter 4

The Data

This chapter will explain the format of the data that we get from the cars and from the loop detectors. It will also discuss the format of the incident database. The directory structure that the **fsp** program expects the data to be in is discussed in Chapter 6. The first section of this chapter will discuss how to download the data and where to place it.

4.1 The Study

In order to estimate the delay savings attributable to the FSP, data was collected over two time periods: once when the FSP was not in operation (the "before" period) and once when the FSP was in operation (the "after" period). The before study took place from February 16 through March 19, 1993 with the after study taking place from September 27 through October 29, 1993. All of the data was collected on a section of the I-880 freeway in Hayward, California as shown in Figure 4.1. The study section was 9.2 miles long and varied from 3 to 5 lanes. An HOV lane covered approximately 3.5 miles of the study section. There were several sections that lacked right-hand shoulders and/or left-hand shoulders. Call boxes were installed at approximately 1/4 mile intervals but that data was not used for the evaluation project. Probe vehicle data was collected on the weekdays during the peak periods (6:30 - 9:30 am and 3:30 - 6:30 pm). And loop detector data was collected from 5:00 - 10:00 am and then again from 2:00 - 8:00 pm. For each study period we collected three different types of data: loop detector data, probe vehicle data and an incident characteristics database.

4.2 Downloading The Data

The data that was collected during the Freeway Service Patrol Evaluation Project is available via anonymous ftp from the machine `www-path.eecs.berkeley.edu` at UC Berkeley. Anonymous ftp is a way of downloading files via the Internet without having to use passwords. To connect to the machine that holds the data at UC Berkeley using anonymous ftp you simply type the command:

```
ftp www-path.eecs.berkeley.edu
```

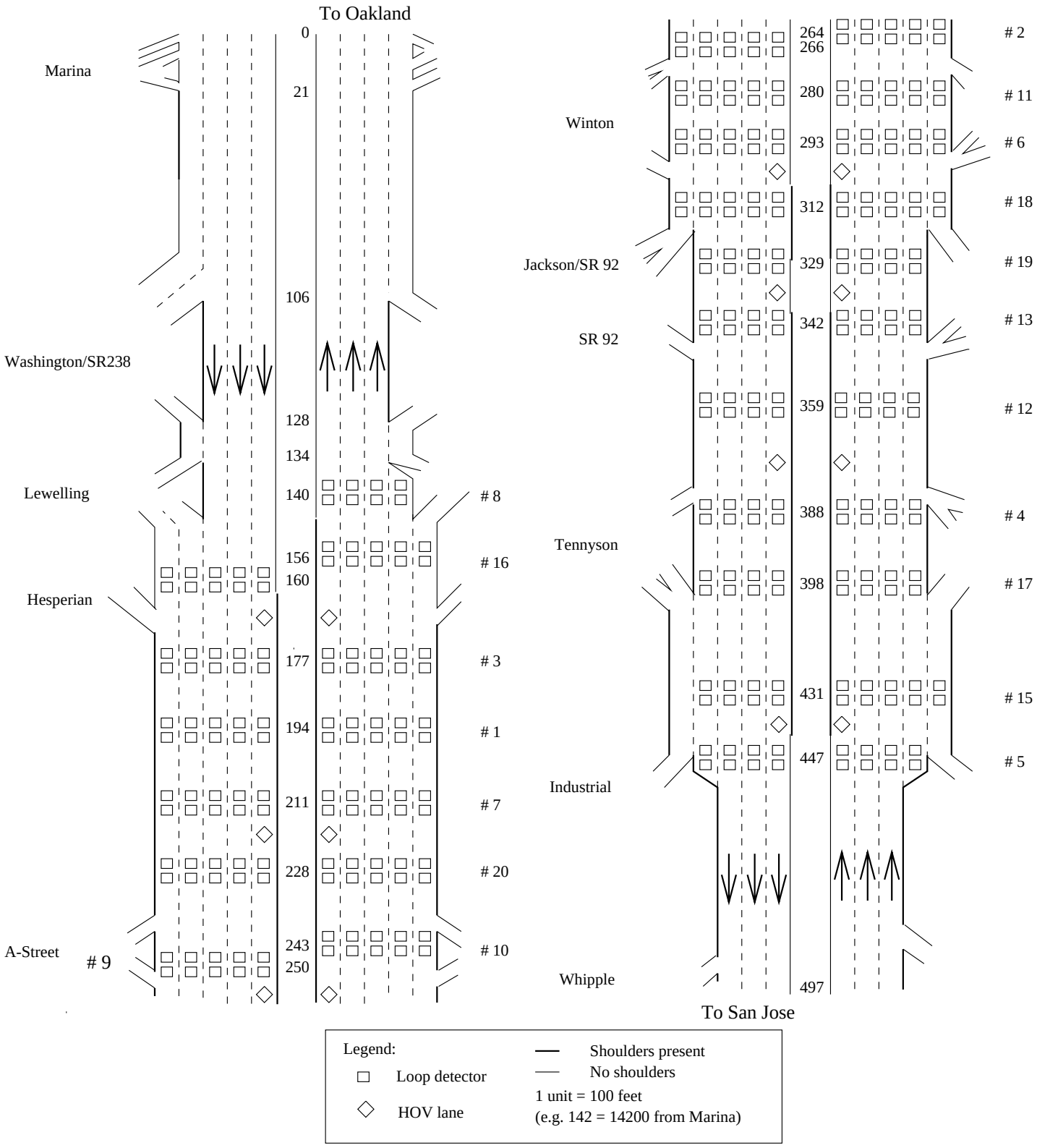

Figure 4.1: The FSP Study Section.

When the machine prompts you for a login ID you simply type in "anonymous." This will tell the machine that you want to log in as a guest. When you are prompted for your password you type in your email address like: `pettyk@eecs.berkeley.edu`.

Once you have connected to `www-path` you will be able to browse through the entire directory structure. One thing that you should always do when browsing through an anonymous ftp site is to download any files named `README` or `README.DOC` because they usually have helpful information in them. After you log in, to get down to the `FSP` directory you need to change into the `pub` directory and then into the `FSP` directory. Once inside the `FSP` directory you'll see that there are three main directories named `Packages`, `Data`, and `Results`. The `Packages` directory holds the support packages for the **xfsp** programs, the `Data` directory holds the data (imagine that), and the `Results` directory holds the results of the project. The discussion here will focus on downloading the data. There are various `README` files scattered throughout the directory structure that can help you download the results and the software packages.

Under the `Data` directory is a sub-directory named `Raw` that holds the raw data, and a sub-directory named `Processed` which holds the processed data. Figure 4.2 gives a representation of what the directory structure looks like for the branch `FSP/Data/Raw/Set2/Loop`. As you can see, the directory names are pretty straight forward.

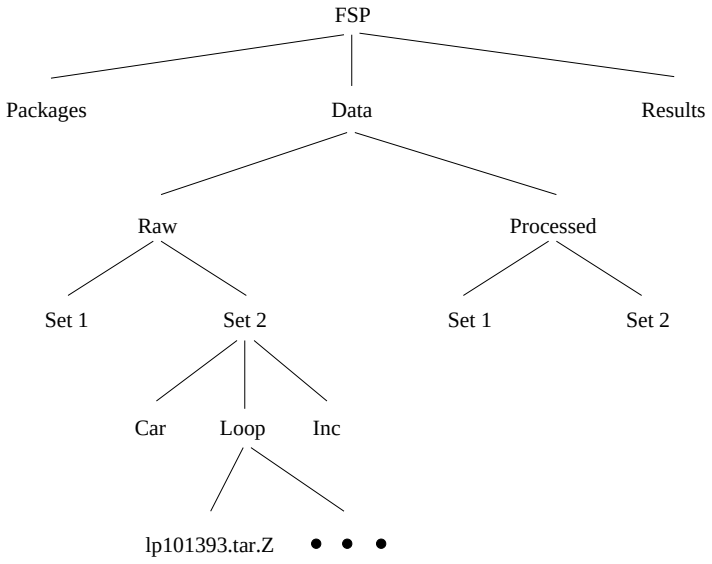

Figure 4.2: FTP Directory Structure On **www-path**.

If you want to run the **fsp** program on the data then you probably want to download the raw data and then use the **fsp** program to process it. In that you case you want to place the data that you download in the input directory structure that is defined in Chapter 6. The data goes in the obvious places: the car data goes in the directory labeled "Cardata," the loop data goes in the directory labeled "Loopdata," etc. If you aren't interested in using the **fsp** program to process the data and you only want to use the data with your own programs then you probably want to download the processed data. The processed data holds, along with the car data and incident database, the loop data in 5 and 1 minutes averages. Most of the processing that has taken place in different groups has been done on the 1 minute loop data

and the incident database.

4.2.1 Downloading Via The World Wide Web

One thing to point out is that you can also download the data through the World Wide Web using the program **netscape** or **mosaic**[1] if you have access to a workstation running X, or by using the program **lynx** if you only have a character based terminal. **Netscape** is a program that allows people to do quite a few things, like browse documents, look at art galleries, listen to music files, and even get current weather maps. What we will use it for is a user friendly interface to download the data. In order for **netscape** to work, it needs to know the location of the document that you want to look through. This is done by specifying something called a uniform resource locator, or URL for short. The URL for the FSP project is:

```
http://www-path.eecs.berkeley.edu/FSP/
```

So to connect to this via **netscape** or **lynx** you would type:

```
netscape http://www-path.eecs.berkeley.edu/FSP
```

or

```
lynx http://www-path.eecs.berkeley.edu/FSP
```

Once you have connect to the FSP document via **netscape** you will be able to download all of the results, the software programs, and whatever combination of the data you would like. I would strongly recommend using this to download the data because it is so user friendly (besides, you'll be surfing the Internet if you do!).

4.2.2 Size of the Data

A word of caution before you start downloading the data: they take up a lot of disk space. Table 4.1 gives the size of each data set in megabytes. Note that if you were to download the entire data set that you would need approximately 2 gigabytes to hold the set uncompressed. To generate any output involving the loop data you will need approximately 400 megabytes more.

	Set 1		Set 2	
Data Type	compressed	uncompressed	compressed	uncompressed
Loop data	382	760	412	817
each day	16	33	16	33
Car data	35	126	31	107
Incident data	0.04	0.2	0.03	0.2

Table 4.1: Size of data sets (in megabyes).

In light of this I would recommend that you download only a portion of the data set and work with that. On the other hand, you can download the processed data, instead of the raw data, and work with that.

[1] On some systems the program **mosaic** is called **xmosaic** or **Mosaic**. Check with your system administrator if you can't find it.

4.3 The Car Data

During the course of the experiment there were four or five probe vehicles that were driven around the study section for approximately 2 1/2 hours in the morning and 2 1/2 hours in the evening. These vehicles were equipped with computers that recorded the car's movement and the driver's key presses and then saved these to various files on a PC floppy disk. We get a total of four files from each car for each run. They are currently named: `key.dat`, `fsp.dat`, `nav.dat`, and `gsp.dat`. Below is a short sample of each type of file and a description of what it is:

key.dat This is a file that saves the keys that the drivers type in.

SAMPLE:

```
 6:54:15   3- 8-93
 0: 0: 2.618        1  14
 0: 0: 5.493        1  7311
 0: 0:21.549        1  03/07/93
 0: 0:44.645       76   06:55:00
 0: 0:56.244       76  qwe
 0:18:16.297    65278  d
 0:21:27.264    81942  k
 0:26:42.116   106153  d
 0:32:53.528   112000  d
 0:36:32.899   132595  l
 0:39:13.795   147085  l
 0:40:29.114   153335  d
 0:41:30.855   158933  k
 0:42:13.116   162578  k
 0:44: 3.463   168739  qwe
```

The drivers type a sequence of keys each time they start a loop. During the before study this sequence was `qwe`. Each time they pass an incident and each time they pass a gore point they type in a single key. The file starts off with a date and time stamp on the first line. This is put there by the computer when it is turned on. The next four lines are just start up information that the user types in. The first main column is the time since the start of the file, the second is the odometer reading of the car in wheel rotations, and the third is the text that the driver has typed in. There is one line in this file for every line the driver types in. A more detailed explanation of the first few lines follows:

```
 0: 0: 2.618        1  14        <- Driver ID number
 0: 0: 5.493        1  7311      <- Car ID number
 0: 0:21.549        1  03/07/93  <- Date
 0: 0:44.645       76   06:55:00 <- Time
 0: 0:56.244       76  qwe       <- Sequence to indicate start of loop
```

```
0:18:16.297    65278  d         <- Key to indicate gore point
0:21:27.264    81942  k         <- Key to indicate an incident
```

In the middle of the experiment it turned out that we needed to have more information from the cars. We wanted a way to get the travel times for the section of freeway that the cars were driving over that was already operating under the Freeway Service Patrol. In order to do this we had to tell the drivers to be more specific when they typed in keys for the file `key.dat`. This means that we had to tell them to type in different keys at different points. These changes can be summarized in Figures 4.3 and 4.4. I will refer to the data taken during the first part of the experiment as the first set and the data taken during the second part of the experiment as the second set.

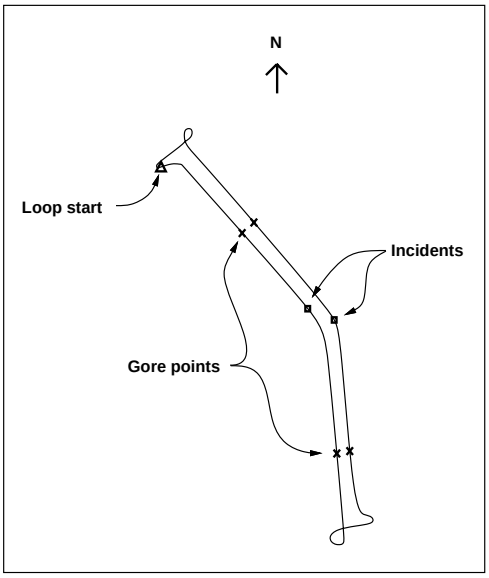

Figure 4.3: Basic Keys Pressed During First Set.

In the first set of data the drivers had to type in something to record three different things happening:

1. Every time they started another loop (or run).
2. Every time they passed one of four gore points.
3. Every time they passed an incident.

To accomplish this they typed in the corresponding keys:

1. Each loop: qwe or QWE.
2. Each gore point: a single key from the left half of the keyboard.
3. Each incident: a single key from the right half of the keyboard.

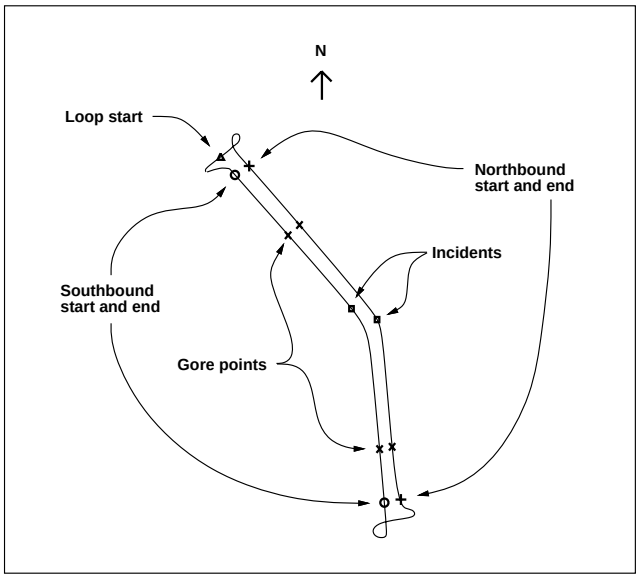

Figure 4.4: Basic Keys Pressed During Second Set.

These keys are labeled in Figure 4.3. In that figure, the boomerang shaped loop is a representation of the freeway. The three different keys that the drivers had to type in are indicated by the three different symbols. We thought that this was going to be a sufficiently rich labeling to capture all of the data that we wanted. It turns out that we needed to have more detail, so for the second set of data, the after study, we had the drivers type in a different set of keys. They typed in a key:

1. Every time they ended another loop (or run).
2. At the start and end of every southbound run.
3. At the start and end of every northbound run.
4. Every time they passed one of four gore points.
5. Every time they passed an incident.

To accomplish this they typed in the corresponding keys:

1. Each loop: the key “q”
2. Southbound run: “c”
3. Southbound run: “t”
4. Each gore point: “n”
5. Each incident: “o”

These keys are labeled in Figure 4.4. Since this is a large number of keys for the drivers to remember to press when they are driving on the freeway, we modified the keyboards to make their task easier. We had hard plastic covers made that fit over the keyboards.

In these plastic covers we cut holes where the desired keys were and placed giant buttons that were clearly labeled on these keys.

You might ask yourself at this point, why am I bringing this up at all? It turns out that in order to take advantage of this added detail, we have to process the car data files differently. In order for the program to be able to do this it has to know what type of data it is dealing with. What this means is that there has to be a way to tell the program whether it is dealing with the first type of data or the second. The way that this is done is through the runfile parameter `CAR_DATA_SET_NUM`. This parameter should be set to a 1 if the first data set is to be used and to a 2 if the second data set is to be used.

fsp.dat This file is saved automatically by the INRAD equipment in the car each time that it drives over an INRAD beacon.

```
SAMPLE:

 6:54:15   3- 8-93
 0:19:19.557    70913  CS4NE
 0:19:19.582    70915  CS4NE
 0:19:19.608    70918  CS4NE
 0:25:37.642   103309  CS2SE
 0:25:37.663   103310  CS2SE
 0:25:37.685   103311  CS2SE
```

The first line is the date stamp (that shows up in all the files). All of the other lines are times when the car picked up an INRAD beacon. The first column is the time since the start of the file, the second column is the odometer reading of the car in wheel revolutions, and the third column is a string to indicate which INRAD signal was picked up. There are a total of three different INRAD points: two on the southbound run and one on the northbound run.

nav.dat This is the data from the digital compass in the car.

```
SAMPLE:
 6:54:15     3- 8-93
      0, 2565, 2640, 2356
      1, 2565, 2640, 2356
      1, 2563, 2641, 2354
      1, 2575, 2639, 2355
      1, 2575, 2639, 2355
      1, 2575, 2639, 2355
```

The `nav.dat` file is a binary file when it is stored on disk. We convert this to it's ascii equivalent which is what is shown above. Once again, the first line is the date stamp. The rest of the rows are stored for each second. The first column is the odometer reading in wheel revolutions, the second and third column is the digital compass reading, and the

fourth column is the angular rate sensor. Although you could calculate the position using the digital compass or the angular rate sensor, we found that the angular rate sensor wasn't very accurate. Therefore we get our position plots from the digital compass.

gps.dat The `gps.dat` file is the data from the GPS equipment in the car.

```
SAMPLE:

 6:54:15   3- 8-93
$GPGGA,025602,3747.75,N,12216.00,W,0,3,000,041,M,-028,M*6E
$GPGGA,025602,3747.75,N,12216.00,W,0,3,000,041,M,-028,M*6E
$GPGGA,025602,3747.75,N,12216.00,W,0,3,000,041,M,-028,M*6E
$GPGGA,025602,3747.75,N,12216.00,W,0,3,000,041,M,-028,M*6E
$GPGGA,025602,3747.75,N,12216.00,W,0,3,000,041,M,-028,M*6E
```

It is stored one line per second, just like the nav.dat file. The first line is the date stamp. The following lines are a bunch of stuff that the GPS equipment stores that we don't really use. Only the third and fifth columns are of use to us. They contain the latitude and longitude of the car which we use to plot the trajectory.

4.4 The Loop Data

The loop data is pretty straight forward in that there is only one file per cabinet per day and it isn't even in ascii text so we can't read it. The loop data consists of the output from different loop detectors. A loop detector is just an inductive loop that is buried under the freeway that picks up the presence of a vehicle traveling over it. On the main line lanes the detectors are placed in pairs, but on the on and off ramps they are single detectors. From this data the program calculates the number of cars that pass over the detectors their average speed and the average occupancy per period. Figure 4.5 should help explain the calculations.

The figure on the left side of Figure 4.5 is a graphical representation of one lane of a freeway. As you can see there is an upstream and a downstream detector and the distance between them, Δ, is a known quantity. The graphs on the right side of Figure 4.5 are hypothetical graphs of the signals that come from the upstream and downstream detectors when a single car passes over both of them. The program finds the difference in time between the falling edges of the two pulses, what is labeled τ in Figure 4.5, and then uses this to calculate the speed. The program calculates the values of speed, occupancy and counts, once per output period. The concept of output period is described in more detail in Chapter 7 but for now it will suffice to call it the frequency at which data is reported to the user. The calculation of speed per output period is as follows:

$$\begin{aligned} \bar{\tau} &= \frac{1}{n}\sum_{i=1}^{n}\tau_i \\ \bar{\nu} &= \frac{\Delta}{\bar{\tau}} \end{aligned}$$

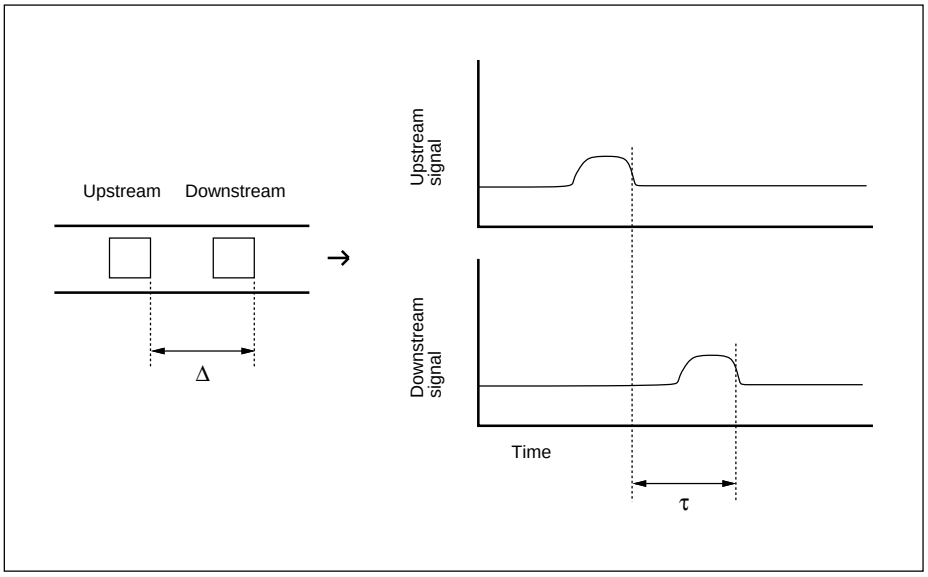

Figure 4.5: Basic Calculation of Car Speeds.

Where n is the number of cars that went over the detectors during that time period, τ_i is the difference in time between the falling edge of the upstream detector and the falling edge of the downstream detector for each car, $\bar{\tau}$ is the average time between falling edges, and $\bar{\nu}$ is the final average velocity. Note that this is the same as the formula:

$$\frac{1}{\bar{\nu}} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{\nu_i}$$

Where ν_i is the speed for each car that went over the detector during the given time period. This is sometimes referred to as the harmonic mean speed. Note that this speed is generated for every lane of the highway and it is not the average speed that we will refer to later on. For a more complete discussion of the speeds used in the delay calculation see Section 11.1.

The occupancy is the percentage of time that the detector has a vehicle on it and the on time is the average time that each vehicle spends on the detector. When the loop data is read in all of these values are calculated. If the user chooses to generate a text file of this data, as explained in Chapter 7, then they will get a file corresponding to the calculated values of counts, occupancy, speeds and on times. What I will explain below is the translation of a bit of loop data into ascii text. This is a sample of the text file from a loop for one output period:

```
This is the data from the file:
/home/clair0/PATH/FSP/Set1/Loopdata/lp021793/loop1.txt

     5:01:00 HACIENDA
           1         2         3         4         5         6         7
PPS     0.51      0.76      2.29      1.27      0.76                0.25
```

```
OCC       0.56      1.47      3.56      1.64      0.85                0.25
ON      166.67    288.89    233.33    193.33    166.67              150.00

PPS       0.51      2.54      2.29      1.27      0.76                0.25
OCC       0.56      2.88      3.64      1.64      1.19                0.28
ON      166.67    170.00    238.89    193.33    233.33              166.67

SPD      71.15      9.30     61.88     63.92     55.79               74.38

            8         9        10        11        12        13        14
PPS       2.54      3.05      4.07      1.53
OCC       3.25      4.52      8.59      2.91
ON      191.67    222.22    316.67    286.11

PPS       2.54      3.05      4.07      1.53
OCC       3.11      4.15      8.50      2.85
ON      183.33    204.17    313.54    280.56

SPD      67.06     63.75     66.70     63.75
```

The above text is an example of one output record for an entire cabinet. This data is for March 17th, 1993 from cabinet #1, at 5:01:00am. Each cabinet can hold a total of 28 loop detectors but most of the time there aren't that many detectors at one site so they hold less. A record like the one above is generated once for every output period. Each record contains 28 slots of data in 2 rows of 14. Each slot contains 2 inductive measurement loops: an upstream and a downstream. You will notice that there are two main rows and in each one of these rows, 2 sub rows. The first sub row is the upstream detector and the second sub row is the downstream detector. I have cut out a section of the above report and put it below with explanations to the right side.

```
            1         2         3        <- Detector number
PPS       0.51      0.76      2.29      <- Upstream PPS value
OCC       0.56      1.47      3.56      <- Upstream OCC value
ON      166.67    288.89    233.33      <- Upstream ON time value

PPS       0.51      2.54      2.29      <- Downstream PPS value
OCC       0.56      2.88      3.64      <- Downstream OCC value
ON      166.67    170.00    238.89      <- Downstream ON time value

SPD      71.15      9.30     61.88      <- Speed value
```

The mapping from the detector numbers to the actual lanes is done by the loop wiring diagram files discussed in Chapter 6. If there is a whole column of values missing then that means that there just isn't any data for that detector slot. The empty columns are just extra.

In some columns there might be data for just the upstream or just the downstream detector. This usually happens when that detector slot holds the output from one of the on or off ramps. If the output from two on or off ramps are placed in the same detector slot - meaning one was placed in the upstream position and one in the downstream position - then there won't be a speed value calculated for this slot. This is because the two detectors might not even be next to each other on the road, and therefore it doesn't make any sense to calculate a speed value.

There is one thing about the way the loop data is processed that I should point out before going on. There used to be two different ways to process the loop data. The first way was called the long data set. This simply meant that we were referring to the whole time period over which the loop data was collected. The second way was called the short data set. This referred to a more specific range of time that the loop data was collected. The original intention was that you would want to have a coarse look at all of the data and then a very fine, detailed look at a smaller section. But we decided that the long data set was not very useful. All of the analysis was being done on the short data set instead of the long data set. Therefore, the **fsp** program no longer has two different ways of processing the data - there is only one.

4.5 The Incident Data

The incident data is one database file with approximately 80 columns of information per incident. The database was collected during the experiment by the drivers of the probe vehicles. When they were driving around the freeway and they passed an incident they would radio it in to the command center (a person sitting at Denny's). Their report would go something like, "There's a stalled green passenger car, southbound, about 1/2 mile before 92. They are getting assistance from the FSP guy now. There is also a CHP on the scene." All of this would be written down by someone at the command center on a standard form. This form was then coded into a database with numerical entries in each column so that a computer could process it later. The database format was devised by Hisham Noeimi and Dan Rydzewski. A complete listing of the columns in the database is given below:

Column	*Name*	*Description and Options*
A	Type:	Data type
		F = Field data
		C = CHP data
		T = Tow truck data
B	Incident:	Incident number
C	Date:	Date incident occurred
D	Shift:	Shift during incident
		0 = AM shift
		1 = PM shift
E	Time:	Time listed in military time
F	Direction:	Direction of incident
		0 = Northbound
		1 = Southbound
G	Beginning:	Incident present at beginning of shift
		0 = No
		1 = Yes
H	End:	Incident present at end of shift
		0 = No
		1 = Yes
I	Link Identity:	Link identity according to between exits
		1 = Marina - Washington/238 intersection
		2 = Washington/238 intersection - Lewelling/Hisperian
		3 = Lewelling/Hisperian - A-Street
		4 = A-Street - Winton
		5 = Winton - Jackson/92/San Mateo Bridge
		6 = Jackson/92/San Mateo Bridge - Tennyson
		7 = Tennyson - Industrial
		8 = Industrial - Whipple
J	Location:	Location listed according to following
		1 = Marina
		2 = Washington/238 intersection
		3 = Lewelling/Hisperian
		4 = A-Street
		5 = Winton
		6 = Jackson/92/San Mateo Bridge
		7 = Tennyson
		8 = Industrial
		9 = Whipple
K	Relative:	Relative location
		0 = At exit
		1 = Before
		2 = After

Column	*Name*	*Description and Options*
L	Exit Distance:	Distance of incident from specific exit
		1 = Right at over/under-pass
		2 = < 1/4 mile
		3 = 1/4 mile
		4 = 1/2 mile
		5 = 3/4 mile
		6 = 1 mile
		7 = > 1 mile
M	Primary Lane:	Which lane was the primary lane
		1 = Lane 1
		2 = Lane 2
		3 = Lane 3
		4 = Lane 4
		5 = Lane 5
		7 = Right shoulder
		8 = Center divide
N	2nd Lane:	Which was the 2nd lane involved
		1 = Lane 1
		2 = Lane 2
		3 = Lane 3
		4 = Lane 4
		5 = Lane 5
		7 = Right shoulder
		8 = Center divide
O	3rd Lane:	Which was the 3rd lane involved
		1 = Lane 1
		2 = Lane 2
		3 = Lane 3
		4 = Lane 4
		5 = Lane 5
		7 = Right shoulder
		8 = Center divide
P	Incident Type:	Type of incident
		0 = Vehicle
		1 = Debris/Pedestrian
		2 = Sweeping/Clearing Debris
Q	Type 1:	Is this a breakdown
		0 = Not this type
		1 = Flat tire
		2 = Gas
		3 = Mechanical
		5 = Can't tell/Using call box

Column	*Name*	*Description and Options*
R	Type 2:	Is this an accident
		0 = Not this type
		1 = single car incident
		2 = multiple car incident
S	Type 3:	Is this a CHP event
		0 = Not this type
		3 = CHP is incident
		4 = Ticketing
T	Begin/End:	When did the incident start and end
		0 = Started and ended during the shift
		1 = Everything else
U	Num Vehicles:	The number of vehicles involved
V-X	Vehicle Type:	Type of 1st, 2nd and 3rd vehicles
		0 = No vehicle involved
		1 = Standard car
		2 = Pickup truck
		3 = Van
		4 = Station wagon
		5 = Motorcycle
		6 = Vehicle with trailer
		7 = Dump truck/Commercial truck
		8 = 18-Wheeler tractor trailer
		9 = Caltrans construction vehicle
		10 = Other
		11 = Tow truck
		12 = 4x4 vehicle
Y-AA	Vehicle Color:	Color of 1st, 2nd and 3rd vehicle
		0 = Not a vehicle
		1 = Black
		2 = Blue
		3 = Brown
		4 = Gold
		5 = Green
		6 = Grey
		7 = Orange
		8 = Red
		9 = White
		10 = Yellow
		11 = Beige
		12 = Black and white (CHP)
AB	Ticketed For Tow:	Vehicle ticketed for tow to storage by CHP
		0 = No, not witnessed
		1 = Yes

Column	*Name*	*Description and Options*
AC	CHP:	Arrival of CHP at scene
		0 = CHP does not arrive during shift at incident
		1 = CHP present at beginning of witness
		2 = CHP arrives during shift at scene
AD	Entries In Log:	Number of times incident is entered in time log
AE-BE	Time Entries:	Individual time entries in log
BF	No Tow:	Tow truck did something funny
		0 = Doesn't apply
		1 = Tow truck left without assisting
		2 = Clearance time not known
BG	Main Clear:	Time that main was cleared
BH	FSP Arrival:	Arrival of FSP at scene
		0 = No FSP
		1 = FSP present at first witness
		2 = FSP present during incident
		3 = FSP is incident
		4 = FSP present but another tow truck towed
BI	CHP Arrival:	Time that the CHP arrived at scene
BJ	Tow Truck Arrival:	Time that the Tow Truck arrived at scene
BK	Ambulance Arrival:	Time that the Ambulance arrived at scene
BL	Fire Arrival:	Time that the fire department arrived at scene
BM	CHP Departure:	Time that the CHP departs the scene
BN	Tow Truck Departure:	Time that the Tow Truck departs the scene
BO	Ambulance Departure:	Time that the Ambulance departs the scene
BP	Fire Departure:	Time that the fire department departs the scene
BQ	Comments:	Were there comments written in the log
		0 = No comments
		1 = Comments written in field log
BR	Official:	Number of official vehicles at incident
BS	Non-Official:	Number of non-official vehicles at incident
BT	Tow Truck Response:	Time that the Tow Truck responds
BU	Tow Truck Clearance:	Time that the Tow Truck cleared incident
BV-CW	Headway:	The headway times
CX	Incident Duration:	The duration of the incident
CY	Weather:	The weather during the incident
		0 = Clear
		1 = Partly cloudy
		2 = Cloudy
		3 = Light rain
		4 = Rainy

For the fields that have a time value if there is no time value or the time value is not available then the database entry is simply a "*". The database is stored on the workstations as a tab delimited set of lines. This means that there is one line per incident and the fields are

separated by tabs.

These are all of the data files that we collect. We get 8 car disks and 19 or 20 loop disks per day and one incident database for the whole study period.

Chapter 5

Problems With The Data

There are quite a few things that turned out to be wrong with the data. In most of the cases we attempted to fix these problems by programming the **fsp** program to take care of them. In other cases there is nothing that we could do about the problems and so we just left them. This chapter will explain the problems that we found and what the **fsp** program does to fix them. Note that this chapter does not have a canonical list of which specific piece of data was wrong where. That information can be found in one of the other publications of the FSP project. All of the runfile parameters that are referred to in this chapter are explained in detail in Chapters 7, 10, and 11.

5.1 The Car Data

There were usually 3 or 4 probe vehicles that were driving around the test section of the freeway during the study period. When the drivers would pass an incident they would press a key on their keyboard to record their position and they would radio in their position and the characteristics of the incident to the base station. Most of the problems with the car data arose during these transactions.

5.1.1 Key Presses

Recall that the drivers were supposed to press a certain set of keys when they were driving around the freeway. The keys that the drivers were supposed to press for the after study are given in Figure 5.1.

So a typical sequence of key presses that the program could expecting to see is something like: loop start, southbound start, gore point, gore point, southbound end, northbound start, gore point, gore point, northbound end, etc. Well, quite a bit of the time the drivers either forgot to press the keys or they pressed the wrong key. This causes a few problems:

- It is harder to match up the incident database with the car key presses. This makes it hard to figure out the precise location of the incident. See the discussion in Section 5.3.1.
- If the drivers don't radio in that they see an incident then it could possibly mess up the incident duration. If they were the first or the last driver to pass an incident and they don't report it then the duration of the incident will be too short.

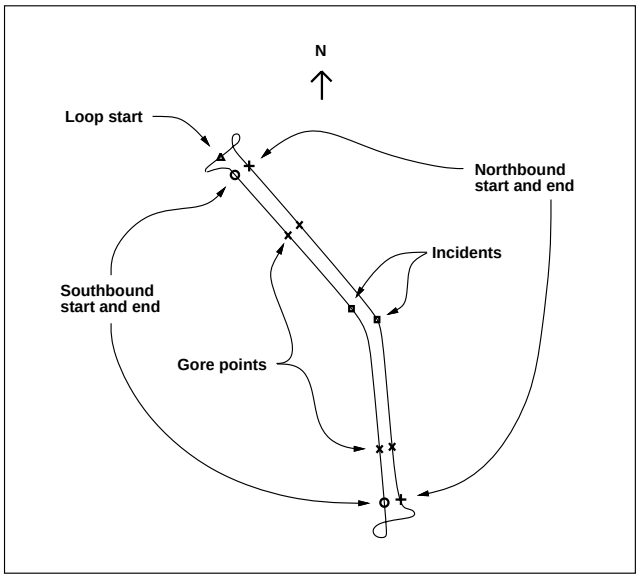

Figure 5.1: Basic Keys Pressed During Second Set.

- In one routine, the program attempts to calculate the travel time for the southbound run and for the northbound run based on the start and end keys. If the drivers don't press these keys properly (in the right order and at the right time) then we can't get accurate travel times.

Although we don't do much to fix the key presses, the program will attempt to weed out simple things. For example, if the driver typed in 1) southbound start, 2) northbound end, 3) southbound end, then the program won't calculate any travel times. What it will still do, no matter what, is parse the car data on the southbound start key. If the driver repeatedly pressed the southbound start key in the middle of a run the program will assume that every key press signified the start of a southbound run. As it turns out, we don't really need to do that much with the key presses for the rest of the analysis to go smoothly.

5.1.2 Car Placement

If the key presses were entered correctly then it's not that hard of a job to figure out where the cars are at any specific time. But you will notice that in the **fsp** program we don't use the car data that much. The reason for this is two fold:

- It was really hard to trust what the drivers typed in. So we couldn't place that much stock in the location of the car at any particular time.
- Getting anything from the car data is just plain hard. If there was ever a way to get any data we needed from the loop data we did.

There was one thing that we did use the car data for, though. That was to refine the placement of each key press in the `key.dat` file. The reason that this had to be done is

because when the incident key was pressed the distance that was stored in the `key.dat` file was the distance since the computer had been turned on. Since the driver may have driven around the block a couple of times before they started their run down the freeway we need to figure out exactly where on the freeway the incident key was pressed. We do this by looking at the other keys that the drivers pressed and at the INRAD data. The procedure that we go through is listed out below:

1. If the INRAD data is available then measure the distance from the last INRAD point to the key press. Since we know where the INRAD points are we can then get the location of the key press.

2. If the INRAD points are not available then look for the other key presses. If you have both a gore point key and a direction starting key then figure out the location of the key press from both of these and average them.

3. If you only have the gore points then just use them.

4. If you only have the direction starting keys then just use them.

Note that this is not the same thing as trying to match up the key presses with the incident database - this is more of a precursor to that routine. We are simply trying to give our best estimate of where the key presses took place based solely on the car data. Trying to match the key presses up with the incident data is another can of worms and is discussed in Section 5.3.1.

5.1.3 Just Plain Bad

Car 2 was consistently bad. It seems that in the middle of operation it would screw up the data stream that it was saving to the `nav.dat` file. We couldn't really understand what it was doing. Maybe the "COM" port on the PC was skipping a byte or something like that. We changed cars in the before study once we figured out that something was wrong. The car was supposed to be fixed for the after study and it wasn't. Since car 2 was always bad, I would recommend that you don't use it for any data analysis.

5.1.4 Car Position Plots

The calculation for the car position from the `nav.dat` files seems to drift. You can see this clearly in Figure 5.2.

The `nav.dat` file contains the output from a digital compass for each second. The output consists of three values, the total distance the car has traveled, and two values whose ratio is proportional to the direction the car is pointing. A typical plot of the car trajectory should have the starting and ending points matching up. Unfortunately, this rarely happens. At first, the reason that we wanted these plots was to make sure that the drivers weren't going to McDonald's in the middle of the experiment. We then decided that this was not important because the drivers were responsible people. So these plots are not used anywhere in the program. These plots are still generated in case somebody wants to do something with them later.

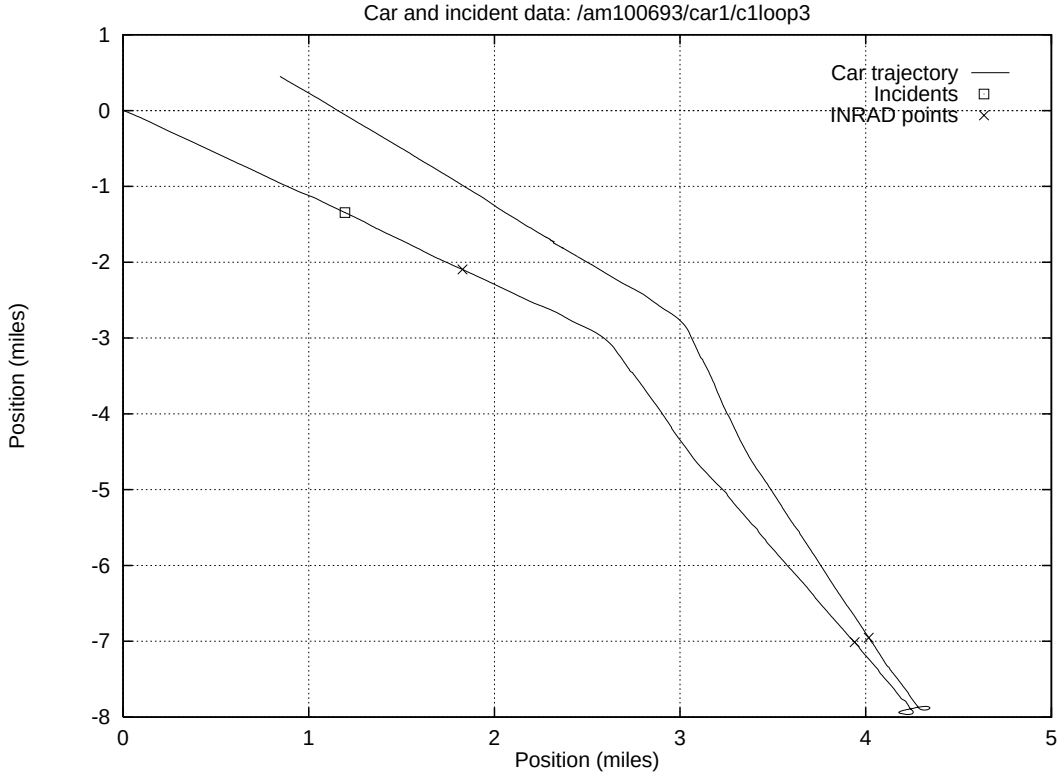

Figure 5.2: Basic Car Position Plot That Drifts.

5.2 The Loop Data

The loop data was our best source of data. Even so, there were still a lot of loop detectors that were continuously out, that went out periodically, or that counted things incorrectly. There are two main fixes that we attempt to do on the loop data: a hole fix, to fill in any missing data, and a consistency fix, to correct systematic errors in the loop data. This section will describe these two fixes and the various problems that we had with the loop data and what we did about them.

5.2.1 Loop Data Drop Outs

There are two different types of missing loop data. The first type is when a loop detector is gone for the whole day - either the computer was broken or the disk was bad, etc. The second type is when the loop detector just doesn't report data for a period of time. If we visualize the data as a plot of detector number vs. time with a solid line if the data is present and a dotted line if the data is not there then we can easily see that we need to fill in the holes.

Figure 5.3 is just such a representation. In this figure the data in loop #7 drops out for a time in the middle of the day and loop #10 is never there. What we do is recreate the data for the missing detectors from the adjacent (adjacent in distance, not time) loop detectors. For example, to recreate the data for loop # 10 we would use detectors # 2 and # 20 somehow.

There are a couple of ways to recreate the missing data. We could just copy the

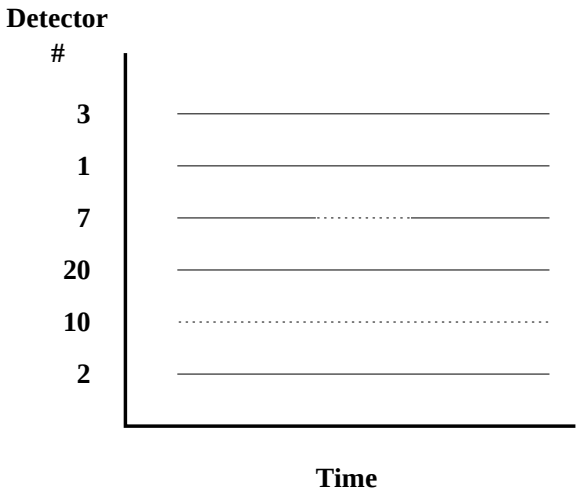

Figure 5.3: Loop Data Dropout.

data from the adjacent upstream loop detector. Alternatively, we could average the values from the two adjacent loop detectors. The criterion that we use to govern how we recreate the data is that the delay should be the same as if the loop detector was not there in the first place. To understand what we are doing here consider two situations:

- In the first situation a loop detector is missing. The adjacent loop detectors expand the length of their segments to cover the missing detector. Based on these new lengths we calculate the delay for these two segments. This is the situation where the loop detector is not there in the first place.
- In the second situation a loop detector is missing but we recreate some data for that detector. We now have three segments all with their original lengths. Based on the new data and the original lengths we calculate the delay for all three segments. This is obviously the situation where the loop data is recreated.

Since these two situations cover the same distance on the freeway the total delay should be the same. This is the criterion that we use to figure out the formulas for recreating any missing loop data. These two situations are displayed graphically in Figure 5.4. The numbers on the top are the loop detector/segment numbers. Note that the segment lengths change from situation 1 to situation 2. The normal equation used for calculating the delay at a particular loop segment is:

$$D_k = L_k \frac{\Delta T}{60} F_k \left(\frac{1}{V_k} - \frac{1}{V_T} \right) \tag{5.1}$$

Where D_k is the delay on segment k, L_k is the length of segment k in miles, ΔT is the time slice in minutes, F_k is the flow on segment k, V_k is the speed on segment k, and V_T is the threshold or congestion speed. So what we are trying to do here is to set the delays for the two situations in Figure 5.4 to be equal. This is represented below:

$$D_{Situation1} = KF_1 \left(L_1 + \frac{L_2}{2} \right) \left(\frac{1}{V_1} - \frac{1}{V_T} \right) + KF_3 \left(L_3 + \frac{L_2}{2} \right) \left(\frac{1}{V_3} - \frac{1}{V_T} \right) \tag{5.2}$$

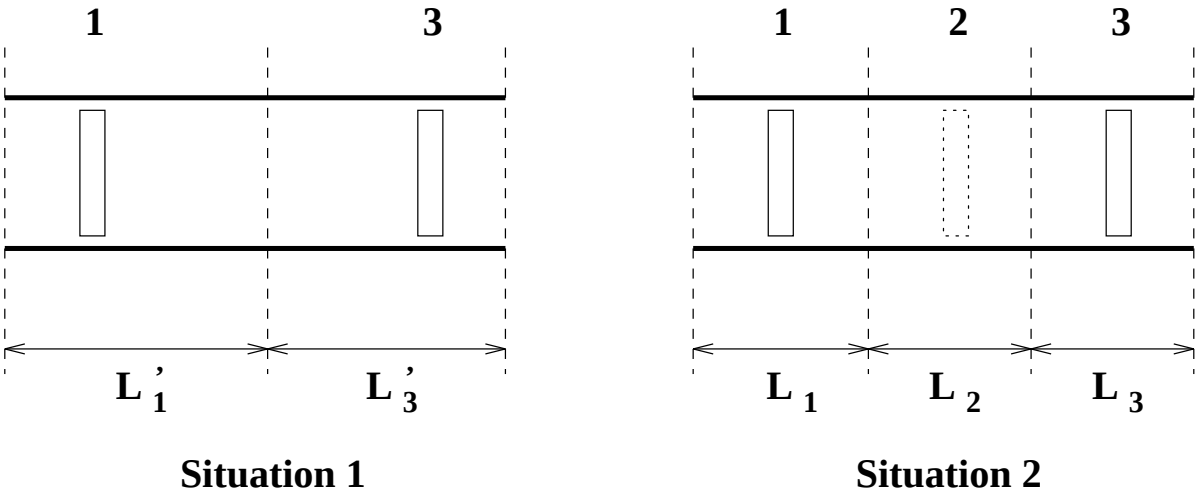

Figure 5.4: Missing Detector.

$$D_{Situation2} = KF_1L_1\left(\frac{1}{V_1}-\frac{1}{V_T}\right)+KF_2L_2\left(\frac{1}{V_2}-\frac{1}{V_T}\right)+KF_3L_3\left(\frac{1}{V_3}-\frac{1}{V_T}\right) \quad (5.3)$$

Where K is the time conversion constant. Setting equations 5.2 and 5.3 equal results in:

$$\frac{F_1L_2}{2}\left(\frac{1}{V_1}-\frac{1}{V_T}\right)+\frac{F_3L_2}{2}\left(\frac{1}{V_3}-\frac{1}{V_T}\right) = F_2L_2\left(\frac{1}{V_2}-\frac{1}{V_T}\right) \quad (5.4)$$

If we assume that this must hold for all values of V_T then we can pull out the terms that have V_T in them to get:

$$\frac{F_1L_2}{2V_T}+\frac{F_3L_2}{2V_T}=\frac{F_2L_2}{V_T} \quad (5.5)$$

Finally, we solve equation 5.5 for F_2 to get:

$$\frac{F_1+F_3}{2} = F_2 \quad (5.6)$$

If we turn around and plug 5.6 into equation 5.4 then we get:

$$\frac{F_1L_2}{2}\left(\frac{1}{V_1}-\frac{1}{V_T}\right)+\frac{F_3L_2}{2}\left(\frac{1}{V_3}-\frac{1}{V_T}\right) = \frac{F_1+F_3}{2}L_2\left(\frac{1}{V_2}-\frac{1}{V_T}\right) \quad (5.7)$$

$$\frac{F_1}{V_1}+\frac{F_3}{V_3} = \frac{F_1+F_3}{V_2} \quad (5.8)$$

$$V_2 = \frac{F_1+F_3}{\frac{F_1}{V_1}+\frac{F_3}{V_3}} \quad (5.9)$$

So the equations that we use to recreate the data for loop detector #2 in situation 2 in Figure 5.4 are equations 5.6 and 5.9 above. Note that the distance drops out and that this will work no matter how many detectors in a row are missing. If the detector that is missing is at the end of our study section, meaning we don't have any data for one side, then the counts, speeds,

and occupancies are simply copied from the side that we do have data for. This option can be turned on or off by the user by specifying the runfile parameter `LOOP_HOLES_FIX` that is discussed in Chapter 7.

Although this is not explained fully until Chapter 11 I feel obliged to mention something. When applied, the loop hole fix will generate a set of loop files that will be referred to as the "gloop" files. The name "gloop" is used to refer to these files because all of them start with the prefix "gloop." This will also be referred to as the second stage in the loop data analysis. The first stage is just the extraction of the raw data from the loop files and the generation of the "floop" files. The third stage is the application of the consistency fix discussed in the next section and the generation of the "hloop" files.

5.2.2 Over/Under Counting

When the loop detectors are setup they are calibrated so that they give consistent results - only one car is counted when only one car goes over the detector, the occupancy time reported is correctly, etc. We noticed that there were some loop detectors that were calibrated incorrectly or that were just broken. In some cases the loop detectors were over counting cars and in others they were under counting cars. This can be a problem for anybody trying to do traffic flow models on the data. A program to identify these problems was written by Kumud K. Sanwal. He ran his program on the loop data and generated a set of correction factors that can be applied to the loop data. When it runs, the **fsp** program can read in these correction factors and fix the loop data such that it is consistent as described in the following subsections. Whether or not this fix is applied is governed by the runfile parameter `LOOP_CONSISTENCY_FIX`. For a complete discussion of the runfile see Chapter 7. The text below explaining the problem was written by Kumud as well.

When applied, the consistency fix will generate a set of loop files that will be referred to as the "hloop" files. The name hloop is used to refer to these files because all of them start with the prefix "hloop." This is the third stage of the loop data analysis.

5.2.2.1 Data

Data is collected from inductive loops that are embedded in the highway. At a specific location, each lane contains a pair of loops that are spaced about 14 feet apart, and during operation, have a current flowing through them. When a vehicle passes over a loop, the effective inductance in the circuit changes and this results in a change in the current flowing in the loop as shown in Figure 5.5. An adjustable current threshold is set for each loop and the current in the loop is compared to this threshold. Binary data is generated by sampling (at 60 Hz) the result of this comparison.

For a desired data output period, the **fsp** program generates the following information for each interval.

- The number of vehicles that pass over the loop location in that interval.
- The fraction of the time in that interval for which the loop is occupied.
- The mean speeds in that region during that interval.

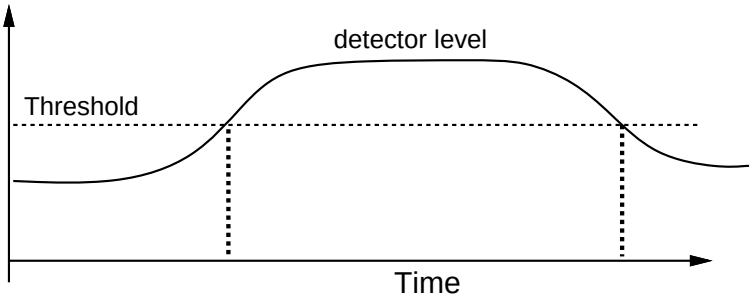

Figure 5.5: Detector Current Level.

5.2.2.2 Sources of Error

The loop data may have inaccuracies/inconsistencies due to a variety of reasons other than a complete loop malfunction (which are taken care of by the **fsp** program). Some of these may lead to errors of the following nature:

Counts: The loop may not be sensitive enough to detect all the vehicles passing over them and thus may in fact be undercounting. Also, vehicles not in the center of the lane may be missed as is the case of vehicles attempting to change lanes. On the other hand, loops may be detecting vehicles that are on neighboring lanes and hence resulting in some overcounting.

Speeds: The resolution with which the time headways can be measured is 1/60 second. This limits the accuracy of measured vehicle speeds. Further, mismatch in tuning of a pair of coupled detectors leads to errors that cannot be compensated for.

Occupancies: The detector thresholds may not be tuned properly and this can result in bias in the measured occupancies.

5.2.2.3 Occupancy Correction

The reliability of the densities obtained from the occupancy data provided by the detectors depends on proper tuning of the detector threshold. In the k^{th} interval, the speed $v[k]$, the flow per lane $q[k]$ and the density $\rho[k]$ are related by

$$q[k] = \rho[k] \times v[k]$$

The traffic density $\rho[k]$ is proportional to the percent occupancy, $occ[k]$, measured by the detector as $\rho[k] = K_d \times occ[k]$, where K_d is a constant that depends on the detector. Substituting this into the previous equation and taking logarithms, we obtain

$$log(q[k]) = log(K_d) + log(v[k]) + log(occ[k])$$

From the field data we can find the least squares regression fit for K_d for each of the main-line detectors. These coefficients can be used to compensate the detector errors by using the estimated constant K_d to obtain the actual density.

5.2.2.4 Counts Correction

Since vehicles are neither created nor destroyed, we can apply conservation laws for vehicles. If we consider a section of highway which has detectors at the beginning and end as well as on the ramps, then the number of vehicles accumulated over the k^{th} data collection interval is

$$\rho[k+1]L - \rho[k]L = q_{in}[k] - q_{out}[k] + r[k] - s[k]$$

If the detectors on a section of the highway count correctly, then the accumulation should not have a drift (should remain bounded within reasonable numbers, the reasonable depending on the length and number of lanes in the section) since this section of highway can only hold a limited number of vehicles. We compute the average accumulation per minute over a long period (about 4 hours) and if its absolute value exceeds a set threshold, then our algorithm checks the consistency of the detectors associated with that section and compensation factors are computed as a fraction of the flow of the nearest mainline flow. Using these correction factors the **fsp** program computes flow estimates that satisfy the consistency requirements.

5.2.3 Bad Initialization

The loop data is collected from 5am to 10am and then again from 2pm until 8pm. When the program reports it's statistics for each time interval (a time period could be 1 min or 5 min, etc.) it reports the number of vehicles, the average occupancy time of the vehicles, and the average speed of the vehicles that went over the detector for that specific time interval. At 5am it turns out that there aren't that many people on the road and it could happen that a whole time interval goes by without a single car going over the detector. If this happens then the loop detector will report the number of vehicles to be zero and the average occupancy to be zero as well - both of these reports are fine. The problem comes when the detectors report the speed for this interval - they will say that the speed is zero. Well, if you are filtering the loop speeds with the loop filtering factor and you come across a speed of zero then this will give you very misleading results. What the program does to correct this is it resets any zero speed from the loop detectors to be the speed from the previous interval. The first couple of samples in the morning might still have zero speed because nothing came before them, but the results are much better. This is not an runfile option - this is on all of the time.

5.2.4 Bad Traps

The loop detector for one lane consists of an upstream and a downstream detector placed about 14 feet apart. Certain pairs of detectors have one of the two traps out, meaning that it always reports zeros. This, of course, causes severe problems:

- The counts for one lane are computed as the average of the two traps. So if one trap is out then the value reported will be only one half of the correct value.
- The speeds will all be zero.
- When calculating the counts and speeds for the whole freeway (meaning the average over all of the lanes) the program will get wrong results.

We take care of this by programming in the configuration file to only look at the trap that works. This tells the program to do a few things:

- Report the counts based only on the good detector.
- Don't use this lane when calculating the average speed.

This option is on all of the time - the user can't turn it on and off. The loop configuration file formats are discussed in Section 6.1.4.

5.3 The Incident Data

The only problems that we had with the incident database was that the location of the incidents was not accurate and the starting and ending times were not accurate.

5.3.1 Bad Placement

It turns out that the location of the incidents is not accurate. The way that we attempt to fix that is by calling a routine that attempts to correlate the locations of the incidents in the incident database with the locations of the key presses in the car data. Our hope in doing this is that we can get a more accurate location for the incident. In the text that follows I will explain what is meant by correlating the data and the limitations and problems that arise in attempting to do so.

What we are trying to do here is to match up two different sets of data: the incident database and the data from the probe vehicles. The incident database was generated from the observations of the drivers. Whenever the drivers would pass an incident they would contact a manager by radio and tell them everything they could about the incident: the type of incident, the number of cars involved, the color of the cars, the location, etc. All of this information was written down and stored in what is now called the incident database. The data from the probe vehicles is basically all the files named `key.dat`. Whenever the drivers of the probe vehicles passed an incident they pressed a key that recorded the location. This information was stored in the `key.dat` file. So we have a `key.dat` file for every car, for every shift, and for every day. This constitutes the car data.

But there are problems with each set of data. In the incident database the location of the incident is only stored in very general terms like, "a half a mile before A-Street," or "3/4 of a mile past Winton." These distances are based solely on the drivers perception and in our experience they are very inaccurate. When we look at the car data the situation doesn't get much better. Even though there was a key that was pressed when the driver passed an incident, there is no link between a specific incident and a key press. There is only a marker in the `key.dat` file that says that a key was pressed when the odometer said a certain value. The odometer reading that is recorded is just the total distance from the start of the shift. So to find out the exact location on the freeway where the key was pressed we need to do a little more processing. This extra processing involves trying to determine where the vehicle was on the freeway based on other keys that the driver typed in. There is a discussion of these problems in Section 5.1. I would like to briefly list out the points that I made above about the two sets of data:

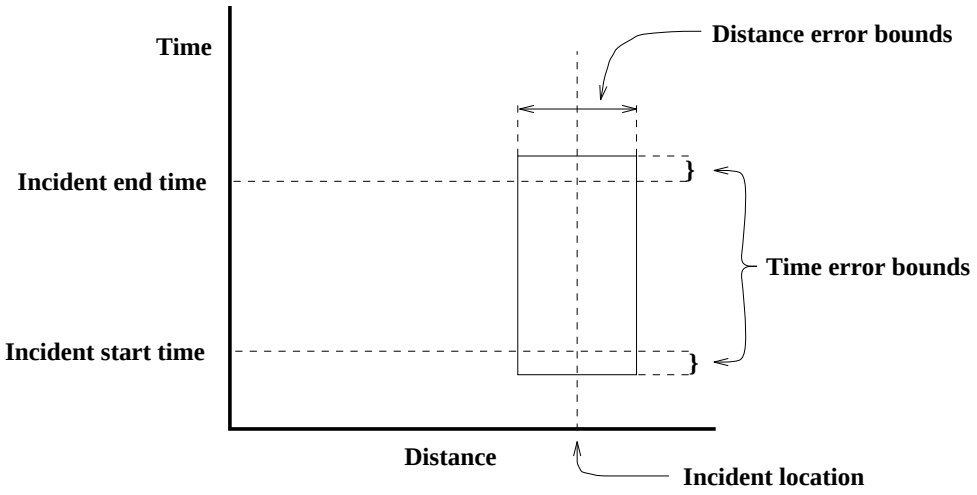

Figure 5.6: Basic Incident Plot.

- Incident Database
  - Includes a lot of descriptive information including the location.
  - The location is very vague.
- Car data
  - A collection of time and distance stamps that should correspond to incidents.

What the correlation routine will attempt to do is to merge the two data sets to get a more accurate incident location. The way that it does this is it first defines a box for each incident in the time-distance plane. This box is centered in distance around the location of the incident recorded in the incident database. The width of the box is the distance error bound that is a function of how accurate we think the data is. The box is centered in time around the recorded time of the incident. The length of the box on the time axis is increased by the amount of the time error bound which is a user settable option. The time error bound is set by the runfile parameter TIME_ERROR_BOUND which is discussed in Chapter 7. Figure 5.6 is a typical picture of an incident plot.

Once the program defines all of the incident boxes on a space-time plot it places all of the key presses from all of the `key.dat` files for that shift and that day on the plot as well. An example of the key presses from the car data is given in Figure 5.7.

The plot on the left in Figure 5.7 is a plot of one run of a probe vehicle down the freeway and back. Since this is a plot of time versus distance, the inverse of the slope is the speed of the vehicle. The steep region in the middle of the run is where the car got off the freeway and turned around. Each "x" is a key press that corresponds to an incident. The plot on the right is just all of the runs for that particular car on one plot. You can see that if you take a vertical line at an incident then it should pass through multiple key presses - these are just the different times that the car passed the same incident.

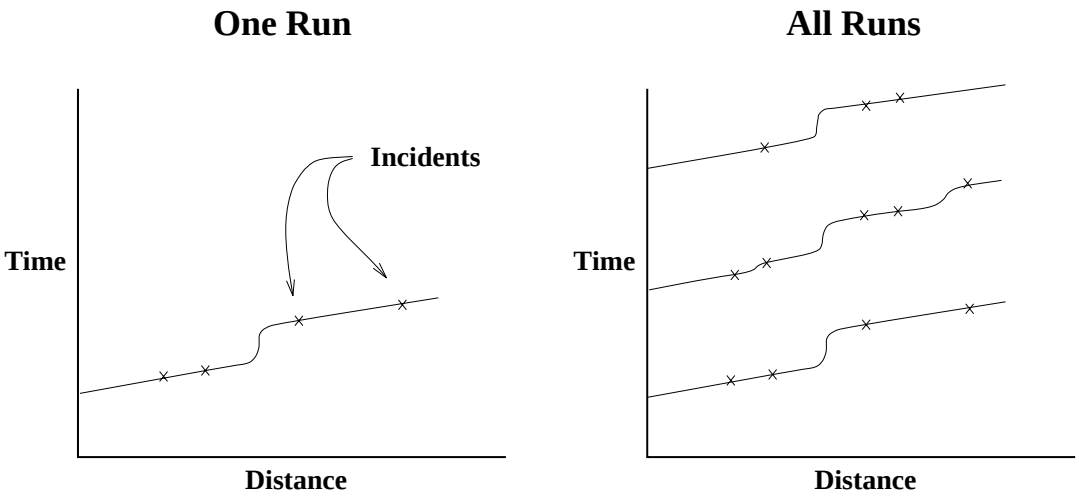

Figure 5.7: Car Trajectories.

Finally, the correlation plot is made from combining all of the incident plots for one shift with all of the key press plots for one shift. This is a lot of information and to make it a little more presentable we take out the trajectory of each car and just leave the key presses. Figure 5.8 is a sample of a correlation plot.

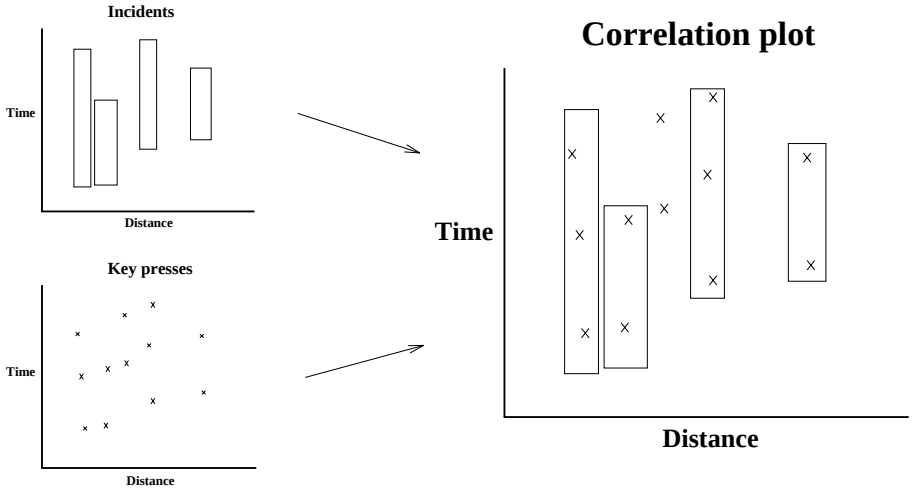

Figure 5.8: Correlation Plots.

Note that the key presses from the key press plots should fall inside of the incident boxes but quite often they don't. Also note that on the final plots the key presses from a specific car are all one symbol:

- car 1: diamond
- car 2: plus
- car 3: square

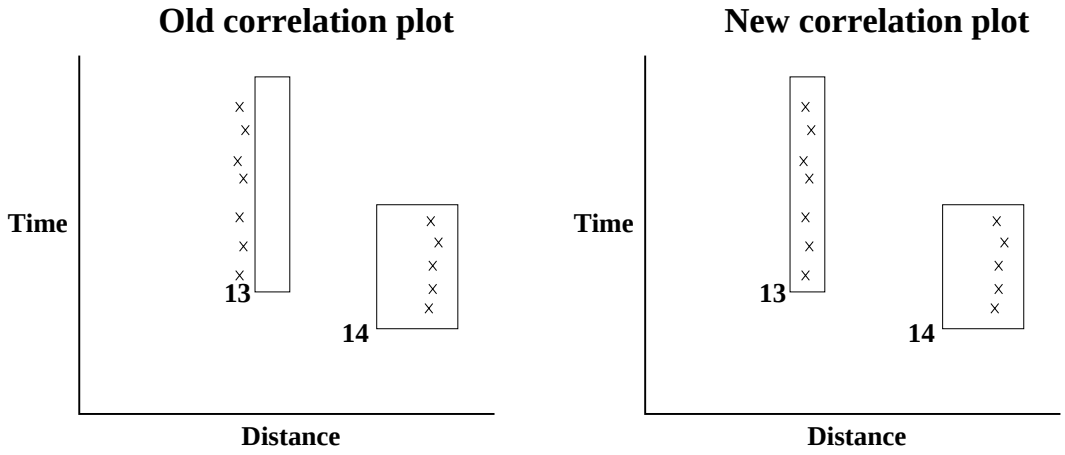

Figure 5.9: Fixed Incident Placement.

- car 4: cross (x)
- car 5: triangle

For an example of a real correlation plot and an explanation of where the files are found see Chapter 16. Once we have made these plots, or in reality, the computer has these plots stored in memory, the program attempts to figure out where the incident actually occurred. It does this by taking the average location of all the key presses that occurred inside of each box. This average location is called the incident location. If an incident doesn't fall within any box then it is not counted at all. There is no attempt to find the closest box that an incident could be in.

It turns out that this process takes a long time. So instead of trying to do this every time that we want to run the program we came up with a different scheme. We took all of these plots and printed them out and looked for places where it was obvious that the placement of the incident was wrong. By wrong we mean there was a whole column of key presses just outside of an incident box. If there is no other incident around then we can be pretty much assured that this incident should be on top of these key presses. We then figured out which direction and how far we needed to shift the incident box for the key presses and the box to match up. Note that we only had to get the key presses inside the box in order for the program to work because the routine will take the average of all of the key presses inside of the box - no matter where they are. We then coded this shift into a file that can be read at runtime to adjust the location of the incidents. Whether or not this file is read in to adjust the incidents is governed by the runfile parameter `FIX_INC_LOCATION` which is discussed in Chapter 7. An example of this process is given in Figure 5.9. So reading in the runtime file and using it to adjust the incidents is a replacement for running the correlation part of the **fsp** program. Of course you can still run the correlation part all that you want. You can even figure out which boxes you want to move around and program those into the incident location fix file. Simply follow the format described in Chapter 6.

Unfortunately the correlation between the incident data and the probe vehicle data

only gave marginal results. If you look at some of the correlation plots the situation looks pretty hopeless. There are many reasons why things could look so bad:

1. The driver could have pressed a key when there was no incident.
2. The driver could have not pressed a key when there was an incident.
3. The computer might have been broken in a car that reported an incident. This might cause an incident to appear in the database that has no corresponding key presses.
4. The location in the incident database could be wrong.
5. The fsp program might not be able to determine the location on the freeway of the key press in the `key.dat` file accurately.

There are a few things that need to be pointed out about the correlation of the incident data:

1. The routine that does the correlation between the incident database and the probe vehicles is turned on and off by the runfile parameter `CORRELATE_CARS_DATABASE`.
2. If this option is not turned on, then the correlation plots will not be made. For an explanation of the correlation plots see the discussion in Chapter 7.
3. This test does not need to be run in order for everything else to work. The only thing that this test does is adjust the location of the incident by looking at the data that the probe vehicles recorded. If you don't want to adjust this value then don't run these tests - everything else will work just fine.
4. You can also adjust the locations of the incidents by reading in the runtime file with the runfile parameter `FIX_INC_LOCATION` discussed above.
5. The only incidents that this test looks at are the ones passed to it from the incident filter. So you aren't going to get a very good correlation if the incidents that you pick from the filter don't overlap with the days that you pick from the car data.

It should be pointed out that what we really need to know is which loop detector was directly upstream of the incident. As you have seen we spend a lot of time trying to adjust the location of the incident and in the end the only granularity that we need is about 1/3 of a mile. Sometimes overkill produces interesting results though.

5.3.2 Bad Duration

Another thing that should probably be corrected in the incident database is the duration of the incidents. Since we only witnessed incidents when a probe vehicle would drive by, we usually don't know when an incident started and ended. The incident duration is obviously longer than the duration that we have in the incident database because those are just times that a car drove by and witnessed the incident. This can be seen in Figure 5.10. In this picture the diagonal lines are the trajectories that the various probe vehicles made. The solid box corresponds to the duration of the incident that we have in the incident database and the dotted line corresponds

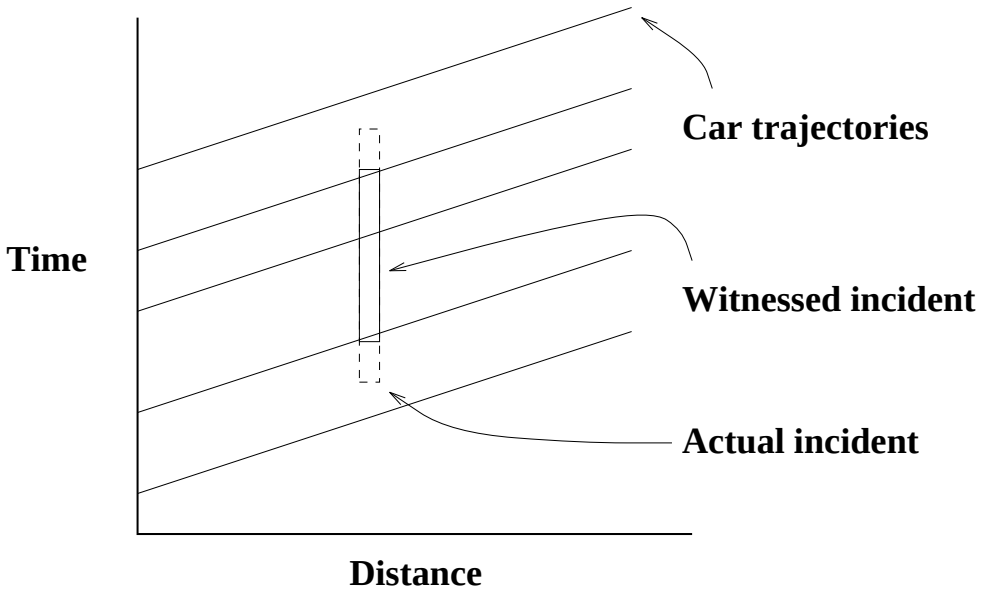

Figure 5.10: Actual Incident and Witnessed Incident.

to the actual duration of the incident. The incident actually occurred sometime between when we witnessed it the first time and when somebody drove by the same spot and didn't witness it. One way to estimate the true incident start and end time is to figure out the average time between any two vehicles and then add one half of this time to the starting time and one half to the ending time. The problem with this is that the headway time is not constant and adding some constant to both sides of the duration might not be the best thing to do. What we could do instead is figure out the last time that a car drove by that didn't witness the incident and then take a certain fraction of this time. The fraction of this time that we usually use is 50% simply because there is no reason to think that start time of the incident isn't uniformly distributed between when we didn't see it and the first time that we did see it. This method should give you a more accurate approximation to the duration of the incident.

Once again, since processing the car data takes a long time we take the familiar route of doing the processing once and then saving the results to a file. This file can then be read in at runtime as a substitution for extracting this information from the car data. The way that this is done is the program will save the last time that a car went by an incident, before it happened and didn't witness it, and the first time that a car went by an incident, after it had ended and didn't witness it. By saving these two values, the user can still choose various values of the fraction of time to use. When the program saves this information, it saves it to a specific file named `inc.duration.out` in the incident data directory. When the program reads this information back in during subsequent runs it reads in the file `inc.duration.in`, also in the incident data directory. So for the program to use the file that it has generated the user has to manually copy the file `inc.duration.out` to `inc.duration.in`. The reason for the two files is to keep the program from overwriting the good data file which took a long time to process (and is very irritating to lose).

Of course, doing this method brings up problems of it's own. Since car 2 was broken we never got any location data from it. But since the driver was still driving around the freeway

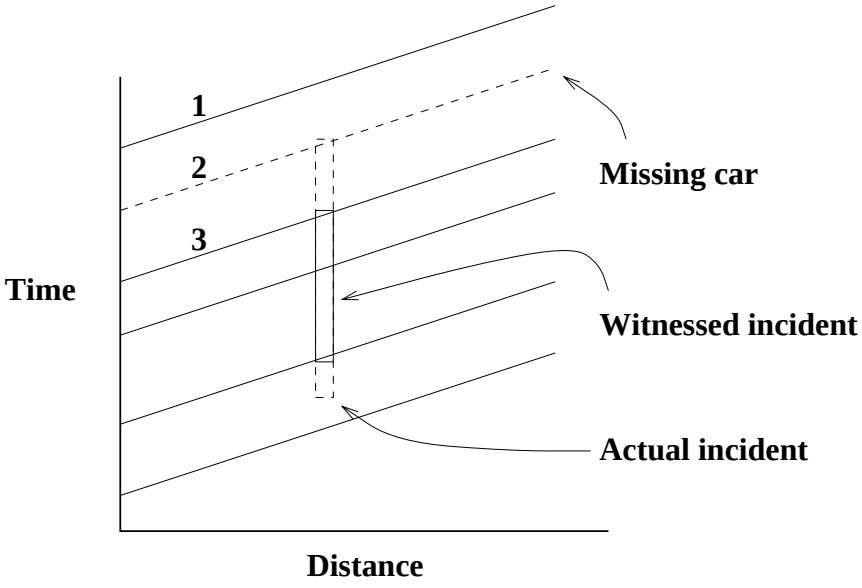

Figure 5.11: Incorrect Incident Duration Fix.

and reporting incidents we have database entries for a car that we have no tach data from. This could easily mess up the duration calculation. For example, let's say that car 3 drove past an incident and witnessed it just as it was being cleaned up, and that the next two cars to pass the incident were car 2 and then car 1 and neither one of them saw the incident. The correct ending time for this incident should be half of the time between when car 3 passed the incident and witnessed it when car 2 passed the incident and didn't witness it. But since we don't have any data from car 2 the **fsp** program thinks that the next car that passed by was car 1. Therefore the ending time for this incident will be erroneously inflated. A picture of this is given in Figure 5.11.

Finally, to be complete, I should mention that another problem that could show up is the fact that we don't know exactly where the probe vehicles are. This is a fuzziness that we always have due to the fact that the vehicle position is dependent on the drivers pressing a certain set of keys at a particular time. As a result we can't be precisely sure of when a car drove by an incident location. I believe, but don't offer any proof, that we can only be accurate to within 1-2 minutes.

Chapter 6

Program Input: Directory Structure and File Formats

The **fsp** program uses two different main directories when it runs: an input directory and an output directory. There are many subdirectories but these are the top ones. All of the raw data is stored under the input directory and all of the output files are placed under the output directory. Since both the input and the output directories are specified by the runfile the output can be placed wherever the user likes by simply modifying the appropriate runfile parameter. The way that the program used to be run is there was only one main directory for both the input and the output. This meant that the output files were placed in the same directories as the data files. There are a couple of disadvantages in doing this. The first is that having one main directory for both the input and the output means that the data files are writable by random users. Since the data is usually valuable and hard to replace it would be best if this was not the case. The second reason that having only one directory is bad is that it makes it difficult for multiple users to work with the data. If all of the output is placed in the same directory then whenever the program is run the output is going to be overwritten. In a multiuser environment where people want to be working with different types of output (5 minute output versus 1 minute output) this is a huge headache. By allowing users to specify the output directory in the runfile they can have the output placed in their own directory and thus avoid having to deal with other users.

So far I have only mentioned the main input and output directories. Underneath each of these directories there are a whole host of subdirectories that the **fsp** program expects to see. The subdirectory structures in the input and output directories are almost the same. The only difference is the contents: the input directories hold raw data and configuration files and the output directories hold processed data. When the **fsp** program runs the first thing that it does is it checks to see if the output directory exists. If it doesn't then the **fsp** program creates all of the subdirectories that it thinks it's going to need. Since the directory structures are almost the same the output directory is sometimes referred to as the parallel directory structure.

6.1 The Input Directory Structure

The **fsp** program expects to see a special input directory structure when it runs. Although some of the directories can be changed by simply changing the appropriate runfile variables, most of the structure is already determined. The **fsp** program also requires the assistance of a few configuration files when it runs. These files must be in the appropriate directories each time the program is run. If you downloaded the software in it's original form and you followed the directions in Chapter 2 then the whole input directory structure will be made for you and all of the configuration files will be put in the correct place. I am including this discussion so that users can understand all aspects of the directories that the **fsp** program uses. What I will describe in this chapter are the different configuration files that are needed and the various input directories that the **fsp** program expects to see. The example that I will use is the file system that I have on my machine. This is illustrated in Figure 6.1.

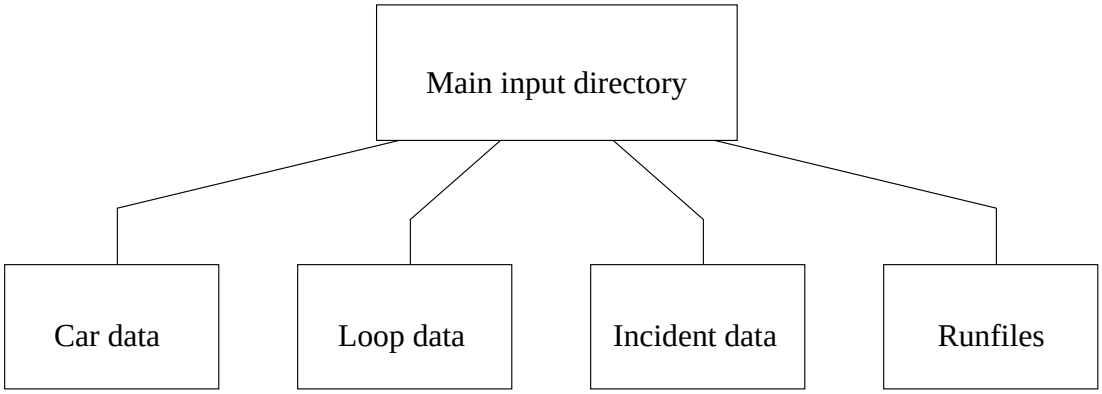

Figure 6.1: Input FSP Directory Structure.

Note that all of the data directories reside under one directory. Also note that the directory labeled "Runfiles" is only used to hold the different runfiles and incident filters that the user might use. The runfiles don't need to be stored here and this directory isn't needed by the **fsp** program. The runfiles are placed here simply for neatness.

6.1.1 Car Input Directory Structure

The car input directory structure is fairly simple. Below the main car directory there is a configuration directory and one directory for each shift of car data. The location of the main car directory is governed by the runfile parameter `CAR_DATA_DIRECTORY`. My directory structure is shown in Figure 6.2.

The directory labeled "Configuration" holds the files needed by the **fsp** program at startup. These files are described below in Section 6.1.2. The directories labeled "day directory" need a little more explanation. Each one of these directories holds one shift worth of data. For example, one directory might hold the morning shift of February 3rd and another one the afternoon shift of February 3rd. Inside each one of these directories are the directories that corresponds to each car that was in operation on that day. These directories correspond to the boxes in Figure 6.2 that are labeled "car directory." The car directories are usually labeled

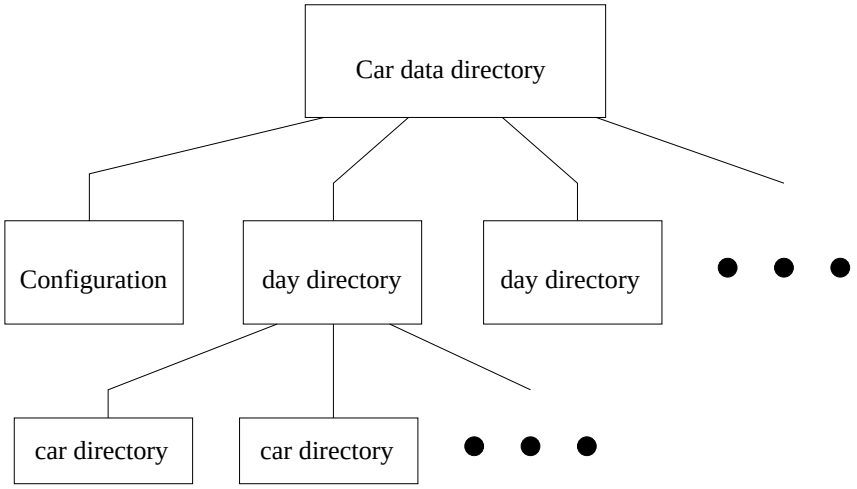

Figure 6.2: Car Input Directory Structure.

"car1," "car2," etc., but this name can be set by the runfile parameter `CAR_DIRECTORY_ROOT`. The car directory structure can be represented with the following picture:

```
am020393            <= This is the day directory
     |
     |-- car1       <= Car directory 1
     |-- car2       <= Car directory 2
     |-- car3       <= Car directory 3
pm020393            <= This is the day directory
     |
     |-- car1       <= Car directory 1
     |-- car2       <= Car directory 2
     |-- car3       <= Car directory 3
```

In the above text there are two day directories, am020393 and pm020393, holding data for the morning and evening shift of February 3rd respectively. Inside of the car directories is the actual data from each car (the four files discussed in Section 4.3).

6.1.2 Car Configuration File Formats

There are two configuration files that the **fsp** program needs in order to interpret the car data correctly. These are the files **DRIVER_FILE_NAME** and **CALIBRATION_FILE_NAME**. Both of these file names are defined in the file `fsp.h`. Below is a short description of each file:

DRIVER_FILE_NAME: This file holds the driver names and their identification numbers. The format is:

```
[ID number]  [First name] [Last name]
```

The ID number can be any unique valid integer and the driver names need to be in two parts. If you don't have a last name then make up one because the program expects there to be two names. When the drivers type in their ID number for the file `key.dat` their name will be matched up with that specific ID. A short sample of the file follows:

```
1       Hisham Noeimi
2       Dan Rydzewski
3       Ayman Taha
4       Jun Huang
```

On my system this filename is defined as:

```
#define DRIVER_FILE_NAME       "drivers.dat"
```

CALIBRATION_FILE_NAME: This file holds the calibration data for each car. The calibration value is the number of wheel revolutions per mile. This value allows the program to calculate the real distance traveled. The format is:

```
[Car number]  [Conversion value]
```

The car number needs to be in the range 1 to **MAX_NUM_CARS** as defined in the file `fsp.h`. If it is not in this range then the program will halt. The value of **MAX_NUM_CARS** is currently defined to be 5. The conversion value should be a valid floating point number. If the program can not find a value for a car then it will print a warning and the program will probably crash later on due to a divide by zero error. An example of this file follows:

```
1       6019.20
2       10032.00
3       6019.20
4       12038.40
5       19588.80
```

On my system this filename is defined as:

```
#define CALIBRATION_FILE_NAME  "calibration.dat"
```

6.1.3 Loop Input Directory Structure

The loop input directory structure is quite a bit like the car directory structure. There is a main loop directory below which lie the directories for the individual days and the configuration files. The location of the main loop directory is governed by the runfile parameter `LOOP_DATA_DIRECTORY`. This structure is shown in Figure 6.3.

The directory labeled "Configuration" holds the files needed by the **fsp** program at startup to interpret the loop data correctly. These files are described below in Section 6.1.4.

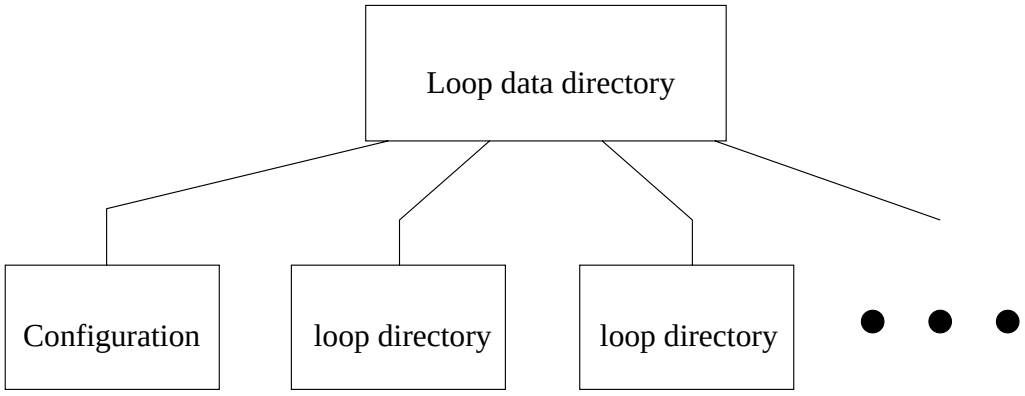

Figure 6.3: Loop Input Directory Structure.

The directories labeled "loop directory" are simply the directories of the various days of loop data. For example, one directory might hold the loop data from February 3rd and another one the data from February 4th. Inside each one of these directories are the data files for each individual cabinet. These are labeled `loop1.dat`, `loop2.dat`, `loop3.dat`, etc. Note that if the files are compressed then they will be labeled `loop1.dat.Z`, `loop2.dat.Z`, etc. This structure can be represented with the following picture:

```
lp020393              <= This is a main loop directory
     |
     |-- loop1.dat        <= loop detector 1
     |-- loop2.dat        <= loop detector 2
     |-- loop3.dat        <= loop detector 3
             .
             .
lp020493              <= This is a main loop directory
     |
     |-- loop1.dat        <= loop detector 1
     |-- loop2.dat        <= loop detector 2
     |-- loop3.dat        <= loop detector 3
             .
             .
```

In the above text there are two main loop directories, lp020393 and lp020493, holding data for February 3rd and 4th respectively.

6.1.4 Loop Configuration File Formats

There are many configuration files that the **fsp** program needs in order to interpret the loop data correctly. These are the display configuration files for the loop cabinets, the wiring diagram files for the cabinet slots, the emission tables, and the geometrics of the road. Below is a list of the various loop configuration files needed:

Display configuration files: These are the files that tell the program how to interpret the loop data when it is first read in and how to display the data when it is printed out. These files came with the original software that Leon Chen wrote and have not been changed since. There are no parameters in these files that can be changed by the user. Therefore I have not listed out their format.

These files have names of the format `loc_ZZ.cnf`. Where "ZZ" is the cabinet number. These files need to be in the directory defined by the variable **LOOPDATA_CONFIG_DIR** in the file `fsp_dirs.h`.

Wiring diagram files: These files are the files that tell the program what slot corresponds to what lane on the freeway. These files have names of the format `loopZZ.wir`. Where "ZZ" is the cabinet number. The format of these files is:

```
[Detector number] [Up or down] [Type] [Lane number]
```

The various legal values are defined below.

Column	*Valid Value*	*Explanation*
Detector:	1 thru 14	The detector number
Up or down:	0	Upstream detector
	1	Downstream detector
Type:	n	Northbound lane
	s	Southbound lane
	f	South off ramp
	F	North off ramp
	p	South passage
	P	North passage
	d	South demand
	D	North demand
	q	South queue
	Q	North queue
	b	Blank
Lane number:	1 thru 6	The lane number
	0	Blank

Any line in the wiring file starting with a "#" is considered to be a comment line and is ignored. Note that since the program wants you to specify what the upstream and downstream detectors are you need to have a line in this file for every detector. Since there are 14 slots, each with an upstream and downstream detector, that means that there needs to be 28 lines in each one of these files. Also note that the characters used to specify the southbound lanes are all lower case while the characters used to specify the northbound lanes are mostly upper case. The one exception is that "n" means northbound lane. A short sample of the file follows:

```
# Loop 16
# Trap Up_or_Down Type Lane
1       0       n       1
1       1       n       1
2       0       n       2
2       1       n       2
3       0       n       3
3       1       n       3
4       0       n       4
4       1       n       4
5       0       n       5
5       1       n       5
6       0       b       0
6       1       b       0
7       0       s       1
7       1       s       1
```

This sample file is `loop16.wir`. Note that this is only the first half of this file. These files need to be in the directory defined by the variable **LOOPDATA_CONFIG_DIR** in the file `fsp_dirs.h`.

Emission Tables: These files hold the tables that are used to do the emission calculations. They are basically tables of the amount of a certain type of emission in grams per hour versus vehicle speed. There is a table for hydrocarbons, nitrogen compounds, and carbon monoxide. The emissions file for carbon monoxide is given below:

```
# This is the table for the emissions of carbon monoxide.
# The values are in grams/mile except for the idle speed
#   which is grams/minute.
# The first row is the idle value for cars, gas trucks, and diesel trucks.
# The second row is the speed values.
# The third through fifth rows are the values for cars,
#   gas trucks, and diesel trucks, respectively, for the
#   speed values in the second row.
3.67 8.81 3.23
5      10   15   20   25   30   35   40   45   50   55   60
44.0  22.5 15.4 11.7  9.3  7.8  6.6  5.8  5.2  4.7  4.3  8.4
105.7 66.4 46.7 35.0 27.6 22.9 20.0 18.3 17.6 17.8 18.8 24.0
38.8  26.7 19.3 14.5 11.5  9.5  8.2  7.4  7.0  7.0  7.3  7.9
```

Any line that starts with an "#" is a comment line and is not read in by the program. You can see that there is a short explanation of the various values in this file in the comments. As it says in the comments, the first non-commented row is the idle value of cars, gas trucks, and diesel trucks respectively in grams/minute. The second row is a list of speed values for which data values will be given later. The third through fifth rows are

the values for cars, gas trucks, and diesel trucks for the speed values given in the second row. Note that for space considerations I have left off a few of the speed values and I truncated the precision of the emissions values.

LOOP_DISTANCE_FILENAME: This file holds the distance in feet of each loop detector from the start of the course. The starting point for the course is the intersection of I-880 and Marina. The distance is always increasing, meaning that the northbound loops are the distance from Marina to Whipple and back to the loop. Note that you need to list out all of the cabinets in the southbound and northbound direction separately - you can't have one entry for a cabinet if it has loop detectors in both the southbound and northbound lanes. You need to list that cabinet out as both a southbound and northbound cabinet. Finally, you have to list out the detectors in order. The specific order that the program is looking for is the order that a car driving around the loop would see if they started at Marina, drove south all the way to Whipple, turned around and drove back to Marina. The format is:

```
[Loop number] [Dist] [Seg length] [South or north] [Number of lanes]
```

Where the loop number is a number from 1 thru 20, and the distance is the distance, in feet, from the starting point of Marina. The segment length is the length, in feet, of the current segment. A segment is the stretch of road covering a loop detector. There is only one loop detector per segment. "South or north" is either **SB** to indicate that this is a southbound detector or **NB** to indicate that this is a northbound detector. Finally, "Number of lanes" is simply the number of main line lanes. An example of this file follows:

```
# Distance Loop Direction #lanes
# These distances were typed in on 6/13
# south bound loops
16      15950   1970    SB      5
3       17700   1880    SB      5
1       19400   1700    SB      5
7       21100   1700    SB      5
20      22800   1550    SB      5
```

Any line that starts with a "#" is treated as a comment line. As you can see by this sample file that the northern most loop detector on the southbound run has to be loop #16. This file needs to be in the directory defined by the variable **LOOPDATA_CONFIG_DIR** in the file `fsp_dirs.h`. On my system this file is defined as:

```
#define LOOP_DISTANCE_FILENAME "loop.distances"
```

6.1.5 Incident Input Directory Structure and Configuration Files

The incident input directory is only one directory. This directory holds the incident database and the configuration files. You should note that there are some files that are generated by

the **fsp** program that can be read in later on to fix the incident data. While these files are read in by the **fsp** if you specify the correct parameters in the runfile they are not technically configuration files so they will not be discussed further. These files are described in more detail in Chapters 5 and 7.

The one true incident configuration file is the file that tells the program what the various fields in the incident database mean. This file is named according to the variable **INCIDENT_DEFS_FILE** which is defined in the file `fsp.h`. This file is usually named `inc_defs.dat`. The format of the file is as follows:

```
[field string] = [numerical value] = [explanation string]
```

The "field string" is the string that is associated with a particular incident database field. These strings are used when creating an incident filter as described in Chapter 9. Since each incident database field has a few different options there is a line in the `inc_defs.dat` file for each option. For example, the incident field that describes the weather has as it's field string "WEATHER." Since there are five different options the section of the `inc_defs.dat` file that describes this field would look like this:

```
WEATHER                 = 0     = Clear
WEATHER                 = 1     = Partly cloudy
WEATHER                 = 2     = Cloudy
WEATHER                 = 3     = Light rain
WEATHER                 = 4     = Rainy
```

Unfortunately, things are not as flexible as they might seem. It turns out that there needs to be a way to tell the program what kind of data each field is: whether it's a time, a date, or a single numerical value. This coding is done within the program itself. This makes changing the format of the incident database something that can only be done by a programmer.

6.2 The Output Directory Structure

As was mentioned above, the output directory structure is quite a bit like the input directory structure. The nicest part about the output directories is that they are created by the **fsp** program at runtime. If you want to place output files in a directory that was used before then that's fine: the **fsp** program will simply write over the old files. If you want to place the output in a new directory then that's fine as well, the **fsp** program will create it for you. The incident and car output directories are exactly the same as the input directories with the exception that there are no configuration directories. The loop output directory structure is only slightly different than the loop input directory structure.

6.2.1 Loop Output Directory Structure

The main loop output directory has a few additional directories that hold various types of output. A graphical representation of the directories is given in Figure 6.4.

The directory labeled "Reports" in Figure 6.4 is the summary directory and it holds the summary loop error reports that are described in more detail in Chapter 15. Note that

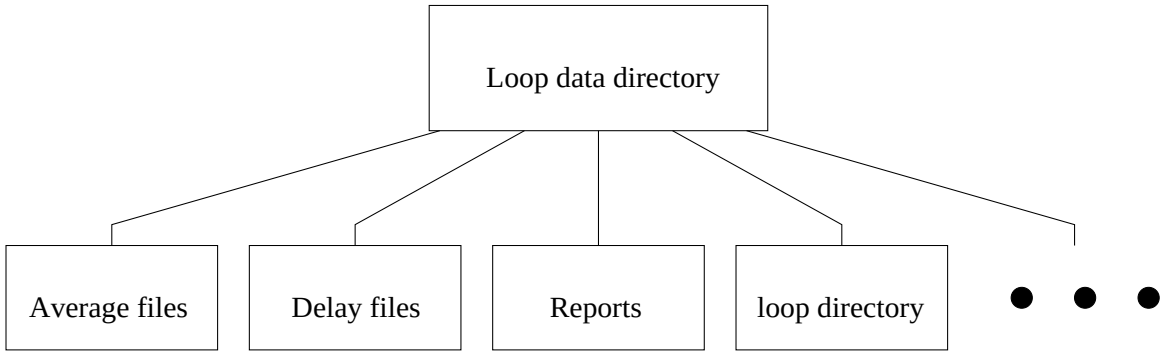

Figure 6.4: Loop Output Directory Structure.

this only holds a certain subset of the loop output text files. The directory labeled "Average Files" contains the average loop files. These files are the average over all of the days of the counts, speeds, occupancies, and densities. These files are needed when calculating the delay with respect to the average. The directory labeled "Delay Files" contains a sub-directory for each main loop directory. These sub-directories contain the delay files for each day with respect to the average. The procedure for computing these files is described in detail in Section 12.7. Finally, the directory labeled "loop directory" holds the output for one specific day.

Chapter 7

Program Input: The Runfile

When the FSP program runs, it takes as one of its arguments the name of a file that is called the runfile. The runfile contains the commands that tell the program what to do: what data to analyze, what tests to run, what output to generate, etc. This chapter is a reference guide to the various parameters that can be specified in the runfile.

There is a whole set of parameters to the runfile that are not listed in this chapter. These are the parameters that deal with the various tests that you can put the loop data through when generating the loop error reports. Since these are all in the same category, and there are quite a few of them, they are described in Chapter 10.

Note that if you are going to be using the program **xfsp** then you don't have to worry about the various names of the runfile parameters because the program will create the runfile for you. You will have to worry about what each parameter does, but the **xfsp** program has on-line help that will replace this chapter. Chapter 8 gives a cross reference guide between the **xfsp** windows and the runfile parameter strings that might be helpful.

7.1 The Basic Idea

I wanted the program to be as flexible as possible and easy to use. Therefore I decided to make the program highly automatic so that if the user has lots of data that they want to crunch through then they can simply create the appropriate runfile and start the program up - everything will be taken care. Unfortunately, the price in software for increased flexibility is increased complexity. In this case the complexity manifests itself in the number of runfile parameters needed to tell the **fsp** program what to do. As you read the rest of this chapter you might conclude that so much flexibility, and hence so many runfile options, just confuses everything. Hopefully, with a little bit of practice, the **fsp** program will become easy to use. Then again, maybe not.

7.2 How To Use The Parameters

The runfile is just a text file that you can edit with any editor. There are a few things about specifying parameters that need to be pointed out:

- To specify a parameter, place it at the start of a new line followed by an equals sign and the chosen value.
- Spaces do not matter. You can have as many spaces as you want between the parameter, the equals sign, and the chosen value.
- If you do not specify a value or you leave out a parameter then the default value will be assigned. A list of the default values is given below in Section 7.4.
- The runfile is case sensitive and all of the parameters have to be specified in upper case.
- If you have a parameter on a line then you must have an equals sign on the line as well. You don't have to specify a value for the parameter after the equals sign, but the equals sign must be there.
- The comment character is the pound sign: #. To comment out a line then just place this character at the start of the line. This is very useful in switching between different parameter options when you don't want to retype a different value every time.
- For most of the parameters you just supply one of a few predefined strings. For some parameters you need to type in your own value. When you do this, just type in your value - don't use quotes, or type casts; the program takes care of all of the casting.
- You can NOT use the logical OR character "|" with the parameters. You can only choose one of the predefined strings.

An example of specifying the parameter DEBUG_LEVEL is given below. Each one of these lines is acceptable.

```
DEBUG_LEVEL = VERBOSE_DEBUG
DEBUG_LEVEL=VERBOSE_DEBUG
DEBUG_LEVEL     = VERBOSE_DEBUG
DEBUG_LEVEL =
#DEBUG_LEVEL = DETAIL_DEBUG
```

In the first three lines the value of the option DEBUG_LEVEL is set to VERBOSE_DEBUG. In the fourth line the value of DEBUG_LEVEL is set to the default value which is SILENT_DEBUG. In the last line the whole line has been commented out. Therefore it is not read by the runfile parser and the value is set to the default value. You could have all of these lines in the same runfile, but the parameter DEBUG_LEVEL would only be set to the last supplied value. Note that in the fourth line above there is no supplied value, so in this case the value would be set to VERBOSE_DEBUG. In the examples below we will use a combination of the different allowable ways to specify a parameter.

7.3 Parameters To The Runfile

What follows is an alphabetical listing of the various runfile parameters. The parameter name is in bold face, flush with the margin, and the explanation is indented.

CAR_CLEANUP Specify whether to leave the temporary car files.

This lets you delete intermediate files that were generated when processing the car data or to leave them hanging around. The files that the program generates when grinding through the car data are completely unimportant and don't hold any useful information like the temporary loop files. This option should only be used if you are trying to debug the program. The options are:

DELETE_FILES: Delete temporary car files.

LEAVE_FILES: Leave temporary car files.

The default for this parameter is **DELETE_FILES**.

CAR_DATA_COMPRESSED Specify whether the car data is compressed or not.

Since the data for this project takes up so much room it is sometimes advantageous to store the raw data in the compressed format. This option will tell the program to expect the data to be in the compressed format or the uncompressed format. The compressed format is the standard UNIX compression format that is done by the programs **compress** and **uncompress**. There are a couple of things to note about using this option:

- You need to have the programs **compress** and **uncompress** in your path. If you type the command **which compress** then it should list out where the program is located. If the program is not in your path then ask your system administrator where it resides.
- All of the data needs to be compressed or uncompressed. You can't have some of the data compressed and some of the data uncompressed.
- If you are only going to be working on a small set of data but you are going to be running the program over and over again then you should probably just uncompress the data before you start running the program - it will take too long for the program to do it for you.
- This option is only useful if you are going to be working with a very large data set and you only want to do the processing once.
- The process of uncompressing and compressing the car data can take up to 10 minutes for each day. So be warned that the processing time for the whole data set might take a while!

The various options are:

DATA_NOT_COMPRESSED: The data is not compressed.

DATA_IS_COMPRESSED: The data is compressed.

The default for this parameter is **DATA_NOT_COMPRESSED**.

CAR_DATA_DIRECTORY Specify which main car directory to use for the data.

This is the complete path of the directory that holds the car data. It must also hold the car configuration and report directories. For more information on the directory structure that the **fsp** program expects see Chapter 6.

The default for this parameter is **/home/clair0/PATH/FSP/Set1/Cardata**.

CAR_DATA_SET_NUMBER Specify whether the data set is from the before study or the after study.

The FSP experiment involved a before and after study of the freeway. Since the data collected from the probe vehicles was different in both studies there needs to be a way to tell the program what type of data set to expect. This parameter tells the program if the car data set being used belongs to the before study or to the after study. For a complete explanation of the difference between the two data sets see Section 4.3. There are two possible values for this parameter, 1 or 2, that correspond to the before and after data sets respectively.

The default for this parameter is **1**.

CAR_DIRECTORY_ROOT Specify the car sub-directory name.

This is the root name to your car directory. The directory structure looks something like the example below:

```
am110492            <= This is a car directory
     |
     |-- car1       <= Sub car dir 1
     |
     |-- car2       <= Sub car dir 2
     |
     |-- car3       <= Sub car dir 3
```

Where the car directory is called am110492, and the various subdirectories below it are called car1, car2, and car3. So the name that we are trying to specify with this parameter is the sub-car-directory name. In this example the name is "car". For more information on the directory structure that the program expects to see then see Chapter 6.

The default for this parameter is **car**.

CAR_SPD_FILTER_FACTOR Specify the filtering factor for the car speed data.

When making the plots for the car data of speed vs. time and speed vs. distance it is sometimes useful to filter the data so as to get a better looking plot. The general formula for filtering data looks like this:

$$\begin{aligned} x_k &= \lambda x_{k-1} + s_k \\ y_k &= (1-\lambda) x_k \end{aligned}$$

Where s_k is the current speed measurement, x_k is the state variable, λ is the filtering factor that you are specifying with this parameter, and y_k is the final output variable. This is simply exponential filtering. Some things to note:

- If the data is filtered then all subsequent analysis will be performed on the filtered data.
- The value for this parameter has to be $0 \leq \lambda < 1$. The program will not let you set the variable to 1 because this results in an unstable filter.
- The larger the value the more smoothing you do. A value of 0 will perform no filtering at all and will give you the actual data.

The default for this parameter is **0.9**.

CORRELATE_CARS_DATABASE Specify whether to attempt to correlate the incident database with the car data.

This tells the program to attempt to correlate the incident database with the data collected from the probe vehicles. We do this to refine the incident location. If you remember, from Chapter 4, when a probe vehicle went past an incident the driver would radio in the location to the base station and they would also press a key on their keyboard. During the radio call the driver would give the distance as something like, "A half mile before 92," or "3/4 of a mile past Winton." These locations are obviously not accurate. Every time that a different probe vehicle would pass the same incident a record would be made at the base station, and in the incident database, of the time that somebody passed it. And when the drivers would press a key, the computer would store the location of the car in a file. So what we have is an incident database entry with a list of times that some probe vehicle passed it and all of the car data files that have times and distances when they passed some incident. What we are attempting to do here is to match up the key presses with the incident log with the ultimate goal of getting a more accurate location for each incident. This process is described in detail in Chapter 5. There are a couple of things to note about this parameter:

- There are two other parameters that effect the output of this routine. These are: INC_CORRELATION_GRAPH and NUMBER_INC_CORR_GRAPHS. The first parameter tells the program whether to make a graph of the keypresses and the incidents and the second one tells the program whether to put the incident numbers on this plot.
- Since this routine is working with car data, you have to specify some car data for this to do anything at all.
- The program has to find the incidents from the car data in order for the correlation to take place. So the parameter INCIDENT_POINTS must be set to YES_INCIDENT_POINTS.
- These tests do not need to be run in order for everything else to work. The only thing that this test does is adjust the location of the incident by looking at the data that the probe vehicles recorded. If you don't want to adjust this value then don't run these tests - everything else will work just fine.

- The correlation of the data take almost as much time as the loop tests. So if you want to run this test then be prepared to wait.

The various options for this parameter are:

NO_CORRELATE: Don't attempt to correlate the data.

YES_CORRELATE: Attempt to correlate the data.

The default for this parameter is **NO_CORRELATE**.

DEBUG_LEVEL Specify how much debug information to print out.

This allows the user to receive various levels of diagnostic output in case there is an error. This probably shouldn't be changed at all because the program is completely free of bugs.... well, maybe. The various options are:

SILENT_DEBUG: No output at all.

MINIMUM_DEBUG: Small updates on progress.

DETAIL_DEBUG: Some updates and more diagnostic output.

VERBOSE_DEBUG: Prints out everything - this will take forever to run.

The default for this parameter is **SILENT_DEBUG**.

DELAY_CALCULATION Specify what type of loop and incident delay calculation to perform.

This parameter tells the program how to calculate the delay for the loop data. The formula for calculating the delay for a particular section of freeway for a specific time is given below:

$$D_k^i = L\frac{\Delta T}{60}F_k^i\left(\frac{1}{V_k^i} - \frac{1}{V_T}\right)$$

Where D_k^i is the delay on segment k during time slice i, L is the segment length in miles, ΔT is the time slice in minutes, F_k^i is the flow on segment k during time slice i, V_k^i is the speed on segment k during time slice i, and V_T is the threshold or congestion speed. What this parameters tells the program is whether to use a constant speed for V_T or to use the average speed. The constant speed that is used, when required, is given by the runfile parameter TRAFFIC_LOW_SPEED. Note that even though this parameter applies to how the loop delays are calculated, it also applies to which loop delay files the program reads in to calculate the delay per incident. A discussion of how to find the average speeds as well as the details on how the delay calculation is performed is given in Chapter 11. The various options for this parameter are:

WRT_CONSTANT_SPEED: Calculate the delay with respect to a constant speed.

WRT_AVERAGE_SPEED: Calculate the delay with respect to an average speed.

The default for this parameter is **WRT_CONSTANT_SPEED**.

DELAY_DOWNSTREAM_NUM Specify how many downstream detectors to use in the incident delay calculation.

There is no consensus on how to calculate the delay for a specific incident. What this option allows the user to do is to choose how many detectors downstream of the incident the program should use in calculating the delay. For each incident the program calculates the duration and location. Then, for the whole duration of the incident the program adds up the delay at each loop detector that is close to the incident. The parameter DELAY_DOWNSTREAM_NUM and DELAY_UPSTREAM_NUM define what is meant by close. This parameter tells the program how many detectors to use downstream of the incident. Normally, one would think that this should be 0 because an incident shouldn't effect anything downstream, only the stuff upstream. Well, as it turns out, you might actually get some benefit downstream from an incident. So we left this decision up to the user. A value of -1 for this parameter means to go all the way downstream to the end of the study section. A more complete description of the delay calculation is given in Chapter 11.

The default for this parameter is **0**.

DELAY_TYPE Specify whether to allow the delay to be negative or not.

When the program applies the formula for calculating the loop delay it is possible for the delay to be negative if the current speed is above the congestion speed. It is a debatable point as to whether or not the delay can be negative. We don't attempt to answer that question but we leave it up to the user. This parameter will tell the program to allow negative delays or to make all negative delays zero. The various options are:

ONLY_HAVE_POSITIVE_DELAY: Allow only positive delays.

HAVE_POSITIVE_AND_NEGATIVE_DELAY: Allow positive and negative delay.

The default for this parameter is **ONLY_HAVE_POSITIVE_DELAY**.

DELAY_UPSTREAM_NUM Specify how many upstream detectors to use in the incident delay calculation.

This parameter tells the program how many loop detectors to use upstream of an incident when calculating the delay for that incident. This parameter is the counterpart to DELAY_DOWNSTREAM_NUM and you should see the discussion there for a short description. For a more complete discussion of the incident delay calculation see Chapter 11.

The default for this parameter is **-1**.

DROPOUT_TIMES Specify whether to figure out the loop data dropout times.

For some reason or another the loop detectors don't work all of the time - they go out sporadically and don't record data. This parameter tells the program to attempt to figure out when those dropout periods occur. It will attempt to do this for all of the data files in each loop directory and it will store the results in one file for that directory. This file is

called `bad.times`. See Chapter 15 for an explanation of the output. If error reports are being generated then the dropout information is included at the end of the error report. If you are going to attempt to fix the holes in the loop data set with the parameter LOOP_HOLES_FIX then you need to set this parameter to YES_DROPOUT_FILE. The various report type options are:

NO_DROPOUT_FILES: Don't attempt to figure out when the data has blackouts.

YES_DROPOUT_FILE: Attempt to figure out when there are data blackouts.

The default for this parameter is **NO_DROPOUT_FILES**.

EMISSION_CALC Specify whether to calculate the loop emissions.

This parameter tells the program whether or not to calculate the emission factors. The emission factors are calculated a lot like the delay values. The formula is given below:

$$E_k^i = L\frac{\Delta T}{60}F_k^i\left(e_{V_k^i} - e_{V_T}\right)$$

Where E_k^i is the emissions on segment k during time slice i, L is the segment length in miles, ΔT is the time slice in minutes, F_k^i is the flow on segment k during time slice i, $e_{V_k^i}$ is the emission factor for speed V_k^i on segment k during time slice i, and e_{V_T} is the emission factor for speed V_T. Note that V_T can be either a constant speed or an average speed as specified by the parameter DELAY_TYPE. There are three different types of emission calculations:

- Carbon monoxide (CO)
- Hydrocarbon compounds (VOC)
- Nitrogen compounds (NITRO)

These are described in more detail in Chapter 11. The various options for this parameter are given below:

NO_CALC_EMISSIONS: Don't calculate any emissions.

YES_CALC_CO_EMISSIONS: Calculate carbon monoxide emissions only.

YES_CALC_VOC_EMISSIONS: Calculate hydrocarbon emissions only.

YES_CALC_NITRO_EMISSIONS: Calculate nitrogen compound emissions only.

YES_CALC_ALL_EMISSIONS: Calculate all emissions.

The default for this parameter is **NO_CALC_EMISSIONS**.

ERROR_FILE_NAME_EXT Specify a new extension for the car error files.

This lets you specify a different name for the extension of the car error files. This could be useful if you want to try out a different error criterion and you don't want to lose the files that you already have. This probably shouldn't be changed.

The default for this parameter is **err**.

FIX_INC_DELAY_BOX Specify whether to use a predefined box to calculate the incident delay.

It turns out that there was quite a bit of debate over how to exactly calculate the delay per incident. This parameter gives the user one more way to calculate this. The plots generated by the parameter INC_CONTOUR_DELAY_PLOT are contour plots of the delay in vehicle-hours. So each line type is a specific level surface. These plots also have the incidents on them as a box. After looking at the delay caused by certain incidents we decided that trying to have the computer automatically figure out the region that contained the delay for each incident was going to be too complex. So we decided that we could define a bounding box for each incident by hand and then code that into a file. This file could then be loaded in at runtime and used by the program to calculate the delay for each incident correctly. A couple of things to note about this option:

- Only certain incidents have a bounding box in the runtime file. This is because it was too hard to figure out what the bounding boxes were for quite a few of the incidents because of overlapping incident.
- You can define more bounding boxes yourself. Just follow the details given in Section 12.11.
- If you tell the program to use the bounding box file then all of the incidents not defined in that file will have a delay of zero.

A more complete description of the calculation of the incident delays is given in Chapter 11. The various options for this parameter are given below:

NO_FIX_INC_DELAY: Don't read in the bounding box file.

YES_FIX_INC_DELAY: Read in the bounding box file and use it to calculate the delay for the incidents in there.

The default for this parameter is **NO_FIX_INC_DELAY**.

FIX_INC_DURATION Specify whether to attempt to fix the incident duration or not.

This parameter tells the program whether to attempt to fix the incident durations or not. Since we only witnessed incidents when a probe vehicle would drive by, we usually don't know when an incident started and ended. The true incident duration is obviously longer than the duration that we have in the incident database because those are just times that a car drove by and witnessed the incident. The incident actually occurred or was cleared sometime between when we witnessed it last and when somebody drove by the same spot and didn't witness it. The way that we attempt to fix the duration is to take the difference between when we witnessed the incident and the last time that a car drove by that didn't witness the incident. We then take a certain fraction of this difference and add it onto the starting or ending time of the incident. This parameter governs whether or not to do this processing and exactly how to do it.

Since the car data takes a long time to process we will try to only do it once. So once we figure out the times from the car data we save them to a file named `inc.duration.out`.

This file will hold two numbers for each incident: the time that a car went by, before the incident occurred, and didn't witness it, and the time that a car went by, after the incident occurred, and didn't witness it. This file can then be read in by the program to substitute for processing the car data. This should save about 4 hours of processing. The one catch is that the file that the program tries to read in is named `inc.duration.in`. So in order for the **fsp** program to use this file after it has generated it the user has to manually copy `inc.duration.out` to `inc.duration.in`. See the discussion in Section 5.3.2 for a more complete discussion of the problems that this generates.

NO_FIX_INC_DURATION: Don't attempt to fix the incident duration at all.

FIX_INC_DURATION_FROM_DATA: This will attempt to fix the incident duration by reading in the car data and matching up the incidents. This routine will only process the car data that has been specified in the runfile. In order for this routine to work you HAVE to have generated the time vs. distance files for the car data. This routine will save the output to a file named `inc.duration.out` that resides in the incident data directory.

FIX_INC_DURATION_FROM_FILE: This will attempt to fix the incident duration by reading in a file that contains the times that cars drove by an incident and didn't witness it. You don't have to do any processing on the car data for this option to work. This file is named `inc.duration.in` and must reside in the incident data directory.

The default for this parameter is **NO_FIX_INC_DURATION**.

FIX_INC_LOCATION Specify whether to fix the placement of the incidents.

One of the things that the program attempts to do is to refine the position of the incidents. The standard way of doing this is to read in all of the car data and to try to match up the key presses in the car data with the incident database, as was discussed under the parameter CORRELATE_CARS_DATABASE. Well, this takes quite a bit of time: someplace on the order of 2 or 3 hours. We have come up with a much faster way of doing this. What we did was first to produce the correlation plots between the incident data and the car data. These plot are generated for each day and each shift and they have all of the key presses that all of the cars made and all of the incidents on them. We then visually lined up where we thought the incidents belonged. Then we coded this into a file which can be read in at runtime to adjust the position of the incidents. This parameter tells the program whether or not to load in this file and adjust the incidents with it. This is much faster than adjusting the position of the incidents based on the car data each time and the results are the same. The various options are:

NO_FIX_INC_LOC: Don't read in the incident placement fix file.

YES_FIX_INC_LOC: Read in the incident placement fix file and adjust the location of the incidents accordingly.

The default for this parameter is **NO_FIX_INC_LOC**.

FLOOP_CLEANUP Specify whether to keep the temporary floop files around.

There are three different stages that the program can go through when it generates the loop data: the floop, gloop, and hloop stages. There are different parameters that determine which of these stages are used. In each stage the program makes some temporary files that can be deleted when the program is done with them. This parameter lets the user decide whether they want the files from a certain stage, the floop stage, to be left on the system. For a complete description of the different stages that the program goes through when calculating the loop data see Chapter 11. A couple of things to note:

- There is basically no reason to leave these files laying around except for debugging purposes or to make sure that the various stages are working correctly.
- The program will not delete files that it might need later on in the program. For example, if you are going to calculate the delay for the incidents then the program will not allow you to automatically delete the last set of loop files, whatever those may be, because they are going used later on. You can still delete them by hand if you wish.

The various options are:

DELETE_EVERYTHING: Delete all of the files from the floop stage.

DELETE_ALL_LANES: When the program generates the loop data for the floop stage it makes a couple of files for each lane in the freeway. This will delete only the lane files.

DELETE_MAIN_LINE_ONLY: The program also generates files that correspond to the average over all of the lanes. This will delete only the average files.

DELETE_RAMPS_ONLY: There are files generated for each on and off ramp as well. This option will delete all of the on and off ramp files.

DELETE_NOTHING: This will leave all of the files from the floop stage.

The default for this parameter is **DELETE_EVERYTHING**.

FSP_DATA_FILE_NAME Specify the name of the fsp data file.

This lets you specify the name of the fsp file that was generated by the computer in the car. This is the file that holds the INRAD data. This probably shouldn't be changed.

The default for this parameter is **fsp.dat**.

GLOOP_CLEANUP Specify whether to keep the temporary gloop files around.

This parameter is basically the same as the parameter FLOOP_CLEANUP except that it deals with the gloop stage. See the discussion under the parameter FLOOP_CLEANUP for a short explanation. See Chapter 11 for a more detailed explanation. The various options are:

DELETE_EVERYTHING: Delete all of the files from the gloop stage.

DELETE_MAIN_LINE_ONLY: The program generates files that correspond to the average over all of the lanes. This will delete only the average files.

DELETE_RAMPS_ONLY: There are files generated for each on and off ramp as well. This option will delete all of the on and off ramp files.

DELETE_NOTHING: This will leave all of the files from the gloop stage.

The default for this parameter is **DELETE_EVERYTHING**.

GNU_PRINTER Specify which printer to have as the default printer for the plotting.

This is the name of the printer to use when generating the gnuplot files. This only works with a UNIX system and when you are generating plots with gnuplot. See Section 13.1 on making plots with gnuplot for a complete explanation.

The default for this parameter is **s307** (the printer in my office).

GORE_POINTS Specify whether to find the gore points or not.

When the drivers are driving around the course they periodically hit a key that signifies that they passed a certain point in the road that we call gore points. When they hit this special key, it is saved in they `key.dat` file with a time and distance stamp. Since there are usually two gore points for each direction, one at the start of the run and one at the end of the run, you can figure out the travel time between the two points. If you choose to figure out the gore points then the program does a couple of things:

- The program picks out all of the gore points from each car.
- It figures out the travel time for each section from the difference between the time stamps of two adjacent gore points.
- It attempts to weed out any mistakes that it detects. This includes multiple hits of gore point keys, missing gore points, and any gore point travel times that are too short. The critical low value is defined in the file **fsp.h** by the defined value GORE_LOW_THRESHOLD. It is currently defined to be 200 seconds. If a travel time is less than this value then the program assumes something is wrong and deletes that gore point.
- It combines all of the valid gore travel times from all of the cars for that shift and makes a plot of travel time vs. run start time for both the south bound traffic and northbound traffic. It places these files in the main car directory. Note that a shift is either morning or afternoon.
- For information on how to make the plots see Section 13.1.

The various options are:

NO_CALC_GORE_POINTS: Calculate points.

YES_CALC_GORE_POINTS: Don't calculate points.

The default for this parameter is **NO_CALC_GORE_POINTS**.

HEADWAY_TIME_VAL Specify the headway time to use.

Since there were only 4 - sometimes 3 - probe vehicles traveling around the study section there is a granularity in the sighting of incidents. The headway time is the average time between any two probe vehicles. This parameter, which must be specified in seconds, allows the user to specify the average headway time. Half of the headway is added to each side of the duration of the incident when calculating the delay. The problem with this is that the variance of the headway is quite large. For a more detailed description of how we calculate the delay per incident see Chapter 11.

The default for this parameter is **420**.

HLOOP_CLEANUP Specify whether to leave the temporary hloop files around.

This parameter is basically the same as the parameter FLOOP_CLEANUP except that it deals with the hloop stage. See the discussion under the parameter FLOOP_CLEANUP for a short explanation. See Chapter 5 for a more detailed explanation. The various options are:

DELETE_EVERYTHING: Delete all of the files from the hloop stage.

DELETE_MAIN_LINE_ONLY: The program generates files that correspond to the average over all of the lanes. This will delete only the average files.

DELETE_RAMPS_ONLY: There are files generated for each on and off ramp as well. This option will delete all of the on and off ramp files.

DELETE_NOTHING: This will leave all of the files from the hloop stage.

The default for this parameter is **DELETE_NOTHING**.

INC_CONTOUR_DELAY_PLOT Specify whether to generate the contour delay plots for the incidents.

We found that a very useful way to visualize the effect of an incident is to generate a plot of distance versus time that has the incident locations marked as well as the contours of delay (or density) that were calculated from the loop data. This parameter lets the user decided if they want to generate these plots or not. If they do want to generate them then one plot is generated for each shift, direction, and day. So for each day there are four plots made. The incidents that are placed on the plots are only the incidents that make it through the incident filter. For a discussion of the incident filter see Chapter 9, and for a discussion of how we use the contour delay plots see Chapter 11. If you want to generate the contour delay plots then the loop data averages need to have been computed. Section 12.7 gives an example of how this is done. The various options are:

NO_INC_CONTOUR_DELAY_PLOTS: Don't generate the contour delay plots.

YES_INC_CONTOUR_DELAY_PLOTS: Generate the contour delay plots.

The default for this parameter is **NO_INC_CONTOUR_DELAY_PLOTS**.

INC_CORRELATION_GRAPH Specify whether to generate the graphs from correlating the car data and the incident database.

This option will tell the program to generate the incident correlation graphs. This is only relevant if the program is already attempting to correlate the incident database with the car data by using the parameter CORRELATE_CARS_DATABASE. If that parameter is not set then this option is ignored.

The correlation graphs are an easy way to visualize what the program is attempting to do when it is trying to correlate the incident database with the car data. A correlation graph is a plot of the incidents from the incident database with the key presses from the car data on a time versus distance plot. This option will generate one correlation graph for each direction and each shift of car data that is specified. The way that these plots are generated is discussed in Chapter 5 and examples of these plots are given in Chapter 11. The various options are:

NO_INC_CORR_GRAPHS: Don't generate the correlation graphs.

YES_INC_CORR_GRAPHS: Generate the correlation graphs.

The default for this parameter is **NO_INC_CORR_GRAPHS**.

INC_DUR_EXPAND_FRACTION Specify what fraction of time to expand the incident duration.

This parameter is used in conjunction with the runfile parameter FIX_INC_DURATION. When we attempt to fix the incident durations we have the time that we last witnessed an incident and the time that a car drove by the same location and the incident wasn't there. This parameter tells the program how much of this time difference to take and add onto the duration. This is usually set to 50% to indicate that we should just take half of the time. Note that the same analysis is done for the starting time and that both the starting and ending times are expanded. See the discussion in Section 5.3.2 for more details. As was discussed in that section, this method of fixing the incident durations gives an estimate of the durations that is biased.

The default for this parameter is **50**.

INC_EXPLANATION Specify what title string to put on the incident graphs.

This parameter is a little different from all of the others in that it is a string. It can be multiple words, it can have commas, etc. The only restriction is that it has to be on a single line and it should probably be less than 50 or 60 characters. The restriction on the length is because it probably won't fit on the graph if it's too long. The graphs that this title string will be placed on are:

- The delay vs duration graph.
- The histogram of the number of incidents vs. duration.
- The histogram of the percentage of incidents vs. duration.
- The cumulative distribution of incidents vs. duration.

The default for this parameter is **Plot** (which is really descriptive).

INC_FINISHED_GRAPHS Specify whether to make a graph of the delay versus duration for the incidents.

This parameter tells the program whether to generate a graph of the delay versus duration for all of the incidents that passed through the incident filter. For an example of this type of graph see Chapter 16. A couple of things to note:

- This will only have an effect if you are already processing the incidents.
- The only incidents that will be plotted are those that passed through the incident filter.
- This does not govern whether the three incident histograms are made. Those are generated whenever you process the incidents.

NO_FINISHED_GRAPH: Don't generate the graph.

YES_FINISHED_GRAPH: Generate the delay vs. duration graph.

The default for this parameter is **NO_FINISHED_GRAPH**.

INC_FINISHED_OUTPUT Specify where you want the finished textual output from the incident analysis to go.

This option tells the program where to put the finished incident output. The finished output is the output that explains the results of the incident database - probe vehicle correlation, the results of the delay calculation for each incident, a table of delay vs incident duration, and various histograms of the incidents. For a discussion of the finished incident output see Chapter 16. The various options are:

NO_FINISHED_OUTPUT: This will simply not generate any output at all.

FILE_FINISHED_OUTPUT: This will place the output in a file.

SCREEN_FINISHED_OUTPUT: This will simply place the output on the screen. This is probably the most useful option.

The default for this parameter is **NO_FINISHED_OUTPUT**.

INC_FINISHED_OUT_LEVEL Specify what level of finished output you want from the incident analysis.

This option tells the program how much to print out about the finished incidents. The finished incidents are simply the incidents that have been processed. There are quite a few different types of output for the finished incidents. For a complete discussion of the output see Chapter 16. The various options are:

INC_FIN_OUT_SPARSE: Don't print out much.

INC_FIN_OUT_MEDIUM: Print out the right amount of information. This is what most people probably want to use.

INC_FIN_OUT_VERBOSE: Print out way too much information.

The default for this parameter is **INC_FIN_OUT_SPARSE**.

INC_GRAPH_MAX_NUM Specify what the y-axis range should be for the histogram of the number of cars vs duration.

This parameter allows the user to set the vertical axis scale on a certain histogram. The reason that we added this parameter is because we noticed that we wanted to be able to visually compare two different plots and that was sort of hard because **gnuplot** usually sets it's axes scales automatically.

The default for this parameter is **10**.

INC_GRAPH_MAX_PERCENT Specify what the y-axis range should be for the histogram of the percentage of cars vs duration.

This parameter allows the user to set the vertical axis scale on a certain histogram. The reason that we added this parameter is because we noticed that we wanted to be able to visually compare two different plots and that was sort of hard because **gnuplot** usually sets it's axes scales automatically.

The default for this parameter is **10**.

INC_MATCH_ZERO_WIDTH Specify whether you want to match incidents that were only witnessed once.

This option tells the program whether it should attempt to match incidents that have zero width. The reason that they have zero width is because they were only witnessed once by one driver, so there is only one entry in the incident database. The program can still calculate a value for the delay caused by this incident because it takes into account the probe vehicle headway. The vehicle headway is a runfile parameter that the user can set that is an estimate of the time between any two probe vehicles. See the text under HEADWAY_TIME_VAL for the relevant discussion. Usually, the incidents that are just witnessed once are small events that aren't really going to contribute to any delays. So this parameter lets you weed those out right off the bat. The various options are:

NO_MATCH_ZERO_WIDTH_INC: Throw out incidents witnessed once.

YES_MATCH_ZERO_WIDTH_INC: Process incidents witnessed once.

The default for this parameter is **NO_MATCH_ZERO_WIDTH_INC**.

INC_RAW_MATCH_OUTPUT Specify where you want the raw output from the incident analysis to go.

This option tells the program where to put the raw incident output. The raw output is the output before any processing is done on the incidents. So it's just a listing of the incidents that made it through the filter. This could be useful if you just wanted to search for different types of incidents and you didn't want to bother with calculating the delays. The various options are:

NO_RAW_MATCH_OUTPUT: This will simply not generate any output at all.

FILE_RAW_MATCH_OUTPUT: This will place the output into a file. For a discussion of the raw incident output see Chapter 16.

SCREEN_RAW_MATCH_OUTPUT: This will simply place the output on the screen. This is probably the most useful option.

The default for this parameter is **NO_RAW_MATCH_OUTPUT**.

INC_RAW_OUTPUT_LEVEL Specify what kind of raw output you want from the incident analysis.

This option tells the program how much to print out about the raw incidents. The raw incidents are simply the incidents that made it through the incident filter. The various options are:

INC_RAW_OUT_SPARSE: Don't print anything. This is probably what most people want to use because the interesting output is in the finished output.

INC_RAW_OUT_MEDIUM: Print some information but not too much.

INC_RAW_OUT_VERBOSE: Print out all the fields of the incident database for each incident that was matched. This is quite a bit of information.

The default for this parameter is **INC_RAW_OUT_SPARSE**.

INCIDENT_DATA_DIRECTORY Specify which main incident directory to use.

This is the complete path of the directory that holds the incident data. This directory must also hold the incident configuration files. For more information on the directory structure that the **fsp** program expects see Chapter 6.

The default for this parameter is **/home/clair0/PATH/FSP/Set1/Incidents**.

INCIDENT_POINTS Specify whether to look for the incidents or not.

This will tell the program to look for the incident points in the car data or not. This option must be turned on if you are going to generate the correlation plots between the incident database and the car data. Since this looks in the car data file `key.dat`, you need to specify some car data for this to have any effect.

NO_INCIDENT_POINTS: Don't record the incident locations.

YES_INCIDENT_POINTS: Record the incident locations.

The default for this parameter is **NO_INCIDENT_POINTS**.

INRAD_POINTS Specify whether to figure out where the INRAD points are.

In each car is an INRAD system that records whenever the car goes over a specific radio beacon in the road. These time stamps are placed in the file `fsp.dat`. This option tells the program to place the INRAD points on the trajectory plots of the cars, and to make a plot of the travel time between INRAD points. Unfortunately, there are only three

INRAD points on the whole route: one at the start of the southbound run, one at the end of the southbound run, and one at the start of the northbound run. So we can only calculate travel times for the southbound direction. Another drawback is that the drivers don't always drive over the INRAD loops in the road, so sometimes we can't get any travel times at all. When we can get the INRAD points we use them to aid in figuring out where the probe vehicle was when the driver pressed a key to indicate that they were passing an incident.

NO_INRAD_POINTS: Don't do anything with the INRAD points.

YES_INRAD_POINTS: Determine the INRAD points and mark them on any relevant plots and generate the travel time plot.

The default for this parameter is **NO_INRAD_POINTS**.

KEY_DATA_FILE_NAME Specify the name of the keyed in file.

This lets you specify the name of the keyed in data file. This file holds all of the key presses that the drivers made while on their runs. This filename probably shouldn't be changed.

The default for this parameter is **key.dat**.

LOOP_AGGREGATE_VALUES Specify whether you want to calculate the aggregate delay and flow.

This option will tell the program to calculate the total delay and flow for all of the loop detectors for a fixed time period. There are two time periods that the program uses: a morning period and an evening period. The morning period is from 6:30am until 9:30am and the evening period is from 3:30pm until 6:30pm. These times correspond to the times that we had probe vehicles on the freeway. We only looked at these times because we wanted to know the aggregate flow and delay only for the times that we knew there were incidents.

To find these aggregate values the program simply reads in all of the loop delay and flow files that have been calculated and sums up the values for the fixed time period. This is done for each set of loop data specified. This will also sum up over all of the loop data sets specified. This should give you a feel for how much delay was taking place over certain days. The various options are:

NO_CALC_AGGREGATE_VALUES: Don't calculate the aggregate delay and flow.

YES_CALC_AGGREGATE_VALUES: Calculate the aggregate delay and flow.

The default for this parameter is **NO_CALC_AGGREGATE_VALUES**.

LOOP_AVERAGE Specify whether to calculate the averages from the loop data.

This option will tell the program whether to calculate the average values for the speeds, counts, occupancies and densities. For each different data type the average is calculated at each time of day with respect to the various days. For example, if you started off with

four days worth of speed data that covered the time period from 8am until 10am in steps of 1 minute then this routine would calculate the average speed over the four days from 8am until 10am in steps of 1 minute. So the average speed value at 8am would be the average of four values, each one corresponding to the speed of the different days at 8am.

These average values are used by the **fsp** program in a few different ways. The first way is in the calculation of the delay with respect to the average. The **fsp** program needs to know what the average speed is before it can calculate the delay this way. For a complete example of how to calculate the averages and to then use them to calculate the delay with respect to the average see Section 12.7. The second way the averages are used is in the generation of the contour delay plots. If the runfile parameter INC_CONTOUR_DELAY_PLOT is set properly then the **fsp** program will generate various contour plots that give a space-time picture of the density around an incident. The contour delay plots are described in more detail in Chapter 16.

There is an average file created for every loop detector for each value. These average files are placed under a special directory named "Avg" that lies under the main loop output directory. These files are named according to the following scheme: **WloopX.YZa**. Where we have the following:

W: is either "g" or "h" corresponding to the gloop or hloop files. The gloop files are the loop files with the holes fixed and the hloop files are the gloop files that have been fixed to be consistent.

X: is the loop number.

Y: is either "n" for northbound or "s" for southbound.

Z: is either "s," "c," "o," or "d," for speeds, counts, occupancies or densities respectively.

For example, the file holding the average speed for fixed northbound loop data at detector #17 would be named `gloop17.nsa`.

There are a few things to note about creating the averages:

- The averages are created for only the speeds (mph), counts (vehicles/lane/hour), occupancies (%), and densities (vehicles/mile).
- The **fsp** program can only make the averages for the loop data with the holes filled in. That means that the loop data file prefix needs to be either "g" or "h" in order for this to work. If you attempt to make the averages for the data with the hole not filled in then the program will crash.
- The **fsp** program will only make the average for the loop data sets that were declared in the runfile. This means that if you are only working with a subset of the data and you tell the program to compute the averages then the program will compute the averages on only a subset of the data - the original averages, if there were any, will be overwritten.

NO_LOOP_AVERAGE: Don't calculate the loop averages.

YES_LOOP_AVERAGE: Calculate the loop averages.

The default for this parameter is **NO_LOOP_AVERAGE**.

LOOP_CONSISTENCY_FIX Specify whether you want to attempt to fix the consistency errors in the loop data.

It turns out that there are certain loop detectors that consistently over counted or under counted the cars going over them. To correct this problem we allow the user to read in a set of predefined files that tell the program how to correct for these consistency errors. This is explained in detail in Section 5.2.2. If this is turned on then this constitutes the third stage in the loop data processing. There are a couple of things to note about the algorithm that does the consistency fix:

- You can only run the consistency fix after the loop hole fix has been run. So you need to set up the program to run the hole fix algorithm at the same time that you want to run the consistency fix. See the discussion under LOOP_HOLES_FIX.
- The consistency fix is only run on the averages across the lanes, not the individual lane files themselves.
- You need to have the consistency fix tables in place in order for this to run. See the discussion in Chapter 6.

The various option are:

NO_FIX_CONSISTENCY_ERRORS: Don't attempt to fix the consistency errors.

YES_FIX_CONSISTENCY_ERRORS: Attempt to fix the consistency errors.

The default for this parameter is **NO_FIX_CONSISTENCY_ERRORS**.

LOOP_DATA_COMPRESSED Specify whether the loop data is compressed or not.

This will allow the user to store the loop data in the compressed format. See the discussion under CAR_DATA_COMPRESSED for the relevant information. The various options are:

DATA_NOT_COMPRESSED: The data is not compressed.

DATA_IS_COMPRESSED: The data is compressed.

The default for this parameter is **DATA_NOT_COMPRESSED**.

LOOP_DATA_DIRECTORY Specify which main loop directory to use.

This is the complete path of the directory that holds the loop data. It must also hold the loop configuration and report directories. For more information on the directory structure that the **fsp** program expects see Chapter 6.

The default for this parameter is **/home/clair0/PATH/FSP/Set1/Loopdata**.

LOOP_HOLES_FIX Specify whether you want to attempt to fix the holes that show up in the loop data.

This parameter will tell the program to attempt to fix the holes in the loop data or not. It does this by figuring out where there is missing data and then making up for it by using the data that surrounds it (i.e. the adjacent loop detectors). This routine will read in the floop files, which are the 1st stage in the loop data processing, and then generate the gloop files, which are the 2nd stage in the loop data processing. Whether the program does the third stage of loop processing is determined by the runfile parameter LOOP_CONSISTENCY_FIX. In order for the hole fix routine to work correctly you need to set a couple of parameters:

- LOOP_FLOW_PLOTS needs to be set to YES_CALC_ALL_FLOW_PLOTS. This tells the program to generate the floop files.
- LOOP_TEXT needs to be set to LOOP_ERR_REPORT_ONLY. This tells the program to process the floop files.
- DROPOUT_TIMES needs to be set to YES_DROPOUT_FILE. This tells the program to figure out where the holes in the data are.
- LOOP_HOLES_FIX needs to be set to YES_FIX_HOLE_ERRORS. This tells the program to attempt to fix the floop files based on what dropout times it was passed. This will generate the gloop files.

So what this routine will do is generate a set of loop files exactly like the floop files except that they are named the gloop files and there shouldn't be any holes in the data. Also note that the gloop files don't have the individual on and off ramp data files. These have been compressed into two file types: the on counts and the off counts. For a complete explanation of the procedure to fix the loop holes see Chapter 5. The various options are given below:

NO_FIX_HOLE_ERRORS: Don't attempt to fix the holes in the loop data.

YES_FIX_HOLE_ERRORS: Attempt to fix the holes in the loop data.

The default for this parameter is **NO_FIX_HOLE_ERRORS**.

LOOP_DIRECTORY Specify a main loop directory to process.

This is one of the two commands that specify data sets for the program to work on (the other one being MAIN_DIRECTORY). This one specifies a loop directory to work on. Just like the car directories, you can have multiple loop directories for each runfile. These directories all need to be subdirectories of LOOP_DATA_DIRECTORY.

The first entry after the equals sign should be the name of the loop directory. After that, the numbers of the loop files for that day should be placed in a row with spaces between each number. An example of specifying several different loop directories is given below:

```
LOOP_DIRECTORY = lp031793 1 2 3 4 5 6 7 8 9 10
LOOP_DIRECTORY = lp031893 1 2 3 4 5 6 7 8 9 10
LOOP_DIRECTORY = lp031993 1 2 3 4 5 6 7 8 9 10
```

In this example there were three loop directories that were specified and all of them contained data for the loop detectors 1 thru 10. The operations that are defined in the rest of the runfile will be performed on all three of these directories. Some things to point out:

- The loop directories have to reside in the directory specified in the runfile parameter LOOP_DATA_DIRECTORY. See Chapter 4 on downloading the data and Chapter 6 on the directory structure for more details.
- Even though this command specifies the loop data set it doesn't tell the program what to do with the data. So if you aren't getting any output then try to specify one of the following commands: LOOP_TEXT, or LOOP_FLOW_PLOTS.

LOOP_FILTER_FACTOR Specify the filtering factor for the loop data.

This is exactly the same principle as the parameter CAR_SPD_FILTER_FACTOR except that this is for the loop data. See the discussion there for an explanation of the algorithm used. This filter will be applied to all of the loop data; the speeds, occupancies, and counts. All analysis and error checking will be performed on the filtered data.

The default for this parameter is **0.9**.

LOOP_FLOW_PLOTS Specify what kind of plots to make from the loop data.

This is one of the options to control what kind of data to extract from the loop data. The other one is LOOP_TEXT. TEXT. The loop data contains occupancies, speeds, and counts for every main line lane, and every demand and passage detector. This option will dictate what gets extracted into various files for generating plots. The output period for the data set is governed by the parameter LOOP_OUTPUT_PERIOD and the time period by LOOP_START_TIME and LOOP_END_TIME. Once the plots are generated they can be viewed on the screen or printed, using the gnuplot program, to the printer specified by the parameter GNU_PRINTER. See Section 13.1 on gnuplot for further information on printing and viewing the files.

The various options are:

NO_CALC_LOOP_FLOW_PLOTS: Don't generate any plots at all.

YES_CALC_OCC_FLOW_PLOTS: Generate only the plots for occupancy.

YES_CALC_SPD_FLOW_PLOTS: Generate only the plots for speed.

YES_CALC_PPS_FLOW_PLOTS: Generate only the plots for PPS or counts.

YES_CALC_ALL_FLOW_PLOTS: Generate all of the plots.

The default for this parameter is **NO_CALC_LOOP_FLOW_PLOTS**.

LOOP_END_TIME Specify the ending time of the loop data set.

This is the ending time, in seconds since midnight, of the loop data.

The default for this parameter is **72000** (8:00pm).

LOOP_OUTPUT_PERIOD Specify the output period for the loop data set.

This is the output time period for the loop data. The output period is the period at which the data is generated. This value has to be between 1 and 3600 seconds. Note that all of the data generated from the loop data will have this period.

The default for this parameter is **60** seconds.

LOOP_START_TIME Specify the starting time of the loop data set.

This is the starting time, in seconds, of the loop data. This time is in seconds measured from midnight the night before.

The default for this parameter is **18000** (5:00am).

LOOP_TEXT Specify what kind of reports to generate from the loop data.

This allows the user to focus in on a detailed section of the loop data and to extract data from it. This is one of the two commands that actually extract information from the loop data. The other one is LOOP_FLOW_PLOTS. With this parameter one can focus in on a specific time by specifying the parameters LOOP_START_TIME, for the starting time, LOOP_END_TIME, for the ending time, and LOOP_OUTPUT_PERIOD for the output period. The output will then be generated for that period of time. Note that these are also the variables that govern over what time period the LOOP_FLOW_PLOTS are generated.

The various options are:

LOOP_NO_REPORTS: No output at all.

LOOP_ERR_REPORT_ONLY: Only print out the error report.

LOOP_TEXT_REPORT_ONLY: Only print out the loop data. Using this will just convert the loop data from raw form to text form.

LOOP_BOTH_REPORTS: Do both reports.

The default for this parameter is **LOOP_NO_REPORTS**.

MAIN_DIRECTORY Specify a main car directory to process.

This is one of the two commands that specify data sets for the program to work on (The other one being LOOP_DIRECTORY). This one specifies a car directory to work on. You can have multiple car directories specified for each runfile, you just need to put a line in the runfile for each main car directory that you wish to process.

Since this is not a parameter that has default values the format of this statement is important. The first entry after the equals sign should be the name of the car main directory. After that, the numbers of the car directories should be placed in a row with spaces between each number. An example of specifying several different main car directories is given below:

```
MAIN_DIRECTORY = am031793 1 3 4 5
MAIN_DIRECTORY = pm031793 1 4 5
MAIN_DIRECTORY = am031893 1 4 5
MAIN_DIRECTORY = pm031893 1 4 5
MAIN_DIRECTORY = am031993 1 3 4 5
MAIN_DIRECTORY = pm031993 1 3 4 5
```

In this example there were six main directories that were specified. All of the operations that are defined in the rest of the runfile will be performed on all of the directories. Some things to point out:

- The main car directories have to reside in the directory specified in the runfile parameter LOOP_DATA_DIRECTORY.
- Since you have to specify which main car directories you are talking about, there is no default value for this parameter. Therefore you have to specify a directory if this parameter is going to be in the runfile.
- Even though this command specifies the car data set it doesn't tell the program what to do with the data. So if you aren't getting any output then try specifying some commands.

NAV_DATA_FILE_NAME Specify the name of the `nav.dat` data file.

This lets you specify the name of the navigation data file. This is the file that holds the data that was taken from the PC in the car. This file should be in the raw data format. This probably shouldn't be changed.

The default for this parameter is **nav.dat**.

NUMBER_INC_CORR_GRAPHS Specify whether to put the numbers on the incident correlation graphs.

This option will tell the program to place the incident numbers on the incident correlation graphs. This is only relevant if the program is already attempting to correlate the incident database with the car data and is trying to make the correlation plots. If those options are not set then this option is ignored.

If this option is set then the program will place the incident number on the lower left hand corner of each incident box. The reason that you might not want to use this option is because when there are a lot of incidents and a lot of key presses it might become really hard to see the numbers. For a more thorough discussion see Chapter 11. The various option are given below:

NO_NUMBER_INC_CORR_GRAPHS: Don't place the numbers on the incident correlation graphs.

YES_NUMBER_INC_CORR_GRAPHS: Place the numbers on the incident correlation graphs.

The default for this parameter is **NO_NUMBER_INC_CORR_GRAPHS**.

OUTPUT_DIRECTORY Specify which directory to use for the output.

This is the complete path of the directory that will hold the output from the **fsp** program. One of the first things that the **fsp** program does when runs is it creates a mirror of the input data directories under the output data directory. Then, all of the output data, meaning the graphs, the text, the error reports, etc., is place in this directory. This is done so that multiple users can process the data without having to worry about messing up other users. The only requirement is that the stem of the output directory must already exist. The stem directory is assumed to be the first directory up from the output directory. So if the output directory is `/home/data/FSP/Output` then the stem directory is assumed to be `/home/data/FSP`. The directory named `Output` may or may not exist. If it does exist then that's fine. If it doesn't then the **fsp** program will create it. For more information on the directory structure that the **fsp** program expects see Chapter 6.

The default for this parameter is **/home/clair0/PATH/FSP/Out1**.

OUTPUT_FLOW_AVG_FACTOR Specify whether the counts value in the loop data is per second or per output period.

When displaying the PPS value from the loop data you can either have this value mean pulses per second, or pulses per period. If you are trying to generate a plot of the volume of cars (eg: cars/lane/hour) then you need this value to be MATCH_OUTPUT_PERIOD. This will make the value of PPS be the number of cars that have passed over the detector in the current time period. For example, if the value of LOOP_OUTPUT_PERIOD was 60, meaning a data point is spit out every 60 seconds, and the value of OUTPUT_FLOW_AVG_FACTOR was MATCH_OUTPUT_PERIOD then the PPS values are the number of counts per output period, which in this case is 60 seconds.

In another example, let's assume that the value of LOOP_OUTPUT_PERIOD was 4 seconds. And that during each of the first two seconds there was 1 car that went over the detector and then during the last two seconds there were no cars that went over the detector. If the value of OUTPUT_FLOW_AVG_FACTOR was PER_ONE_SEC then the PPS value would be 0.5 cars/sec. If the value of OUTPUT_FLOW_AVG_FACTOR was MATCH_OUTPUT_PERIOD then the PPS value would be 2 cars/output period.

Note if OUTPUT_FLOW_AVG_FACTOR is set to MATCH_OUTPUT_PERIOD that when you generate the files for the counts data then the individual lane and the on and off ramp data is in counts/output period but that the file for the aggregate flow on the main line is counts/hour/lane.

The various options are:

MATCH_OUTPUT_PERIOD: Causes the true number of cars that went over the detector in the time period to be displayed as the PPS value. For all of the analysis routines the program expects that this setting will be used.

PER_ONE_SEC: Causes the PPS value to be the number of counts per second, averaged over the output period.

The default for this parameter is **MATCH_OUTPUT_PERIOD**.

PERCENT_DIESEL_TRUCKS Specify the percentage of vehicles on the freeway that are diesel trucks.

This allows the user to specify what percentage of vehicles on the freeway were diesel trucks. This information is used in the calculation of the emission factors. The types of vehicles on the freeway are either passenger cars, diesel trucks, or gas trucks. Since the percentage of diesel trucks and gas trucks are specified in the runfile, the percentage of passenger cars is just whatever is left over.

The default for this parameter is **2.0**.

PERCENT_GAS_TRUCKS Specify the percentage of vehicles on the freeway that are gas trucks.

This allows the user to specify what percentage of vehicles on the freeway were gas trucks. See the discussion under PERCENT_DIESEL_TRUCKS.

The default for this parameter is **3.0**.

PROCESS_INCIDENTS Specify whether to do any processing on the incidents.

This option tells the program whether to process the incidents or not. If you want to do anything with the incidents then this has to be turned on. If this is turned on then the program will read in the incident filter and the incident database and attempt to filter the incidents. Once it has done that it will fix the placement of the incidents if told to do so, and then it will attempt to correlate the incidents with the car data if told to do so. Then the program will calculate the delay for each incident. Whether any of this is displayed to the user is determined by the various incident output parameters.

NO_PROC_INCIDENTS: Don't do any incident processing.

YES_PROC_INCIDENTS: Process the incidents.

The default for this parameter is **NO_PROC_INCIDENTS**.

REPORT_OPTION Specify what type of car data diagnostics to perform.

When the program runs it tries to determine if the car data is valid or not. This parameter tells the program what kind of error reports to generate on the car data. You can make a total of four different types of reports varying from short and not very informative to long and detailed; or you can make all four reports. The various report type options are:

SHORT_REPORT_ONLY: This only lists out what data was collected.

MEDIUM_REPORT_ONLY: Will print out a report about every car listing the driver name, start time, gps data results, etc.

HUGE_REPORT_ONLY: Will print out a very detailed report about every car and every run the car makes.

KEY_REPORT_ONLY: Will print out a very detailed report about the key strokes that the drivers typed in during their runs. This will only be generated if the car data set is of type 2. See Section 4.3 for a discussion of the different car data sets.

EVERYTHING: This makes all of the reports.

The files are saved in the directory "Reports" under the car output directory. For more information on the expected directory structure see Chapter 6. The various files are named with the following scheme: {key,huge,med,sm}ZZZZZ.err. This means that the first few characters are either **key**, **huge**, **med**, or **sm** corresponding to the key, huge, medium, or small report. The Z's correspond to the 3rd through 7th characters of the runfile that was specified for this run. That might seem a little strange but you need to remember that the whole process is under computer control from the data disks to the final reports and this fits into the scheme very nicely. For example, if the runfile was called **rf09230.run**, then the car error reports would be called:

```
key09230.err
huge09230.err
med09230.err
sm09230.err
```

For more information on the naming scheme of the runfiles and the whole process in general see Chapter 17.

The default for this parameter is **EVERYTHING**.

SPEED_DIST_PLOTS Specify whether to make distance vs. time plots of the car trajectory.

This option is exactly like the SPEED_TIME_PLOTS option except that it tells the program whether to generate the distance vs. time plots of the car data or not. If the data is filtered by specifying something for the parameter CAR_SPD_FILTER_FACTOR then the filtered data is plotted. The incidents, from the key presses, are always recorded and plotted on the distance vs. time plot. If the parameter INRAD_POINTS is set to the value YES_INRAD_POINTS then the INRAD points are also plotted. The various options are:

NO_SPEED_DIST_PLOTS: Don't generate the plots.

YES_SPEED_DIST_PLOTS: Generate the plots.

The default for this parameter is **NO_SPEED_DIST_PLOTS**.

SPEED_TIME_PLOTS Specify whether to make speed vs. time plots of the car trajectory.

This option tells the program whether to generate the speed vs. time plots of the car data or not. If the data is filtered by specifying something for the runfile parameter CAR_SPD_FILTER_FACTOR then the filtered data is plotted. The incidents, from the key presses, are always recorded and plotted on the speed vs. time plot. If the parameter INRAD_POINTS is set to YES_INRAD_POINTS then the INRAD points are also plotted. The various options are:

NO_SPEED_TIME_PLOTS: Don't generate the plots.

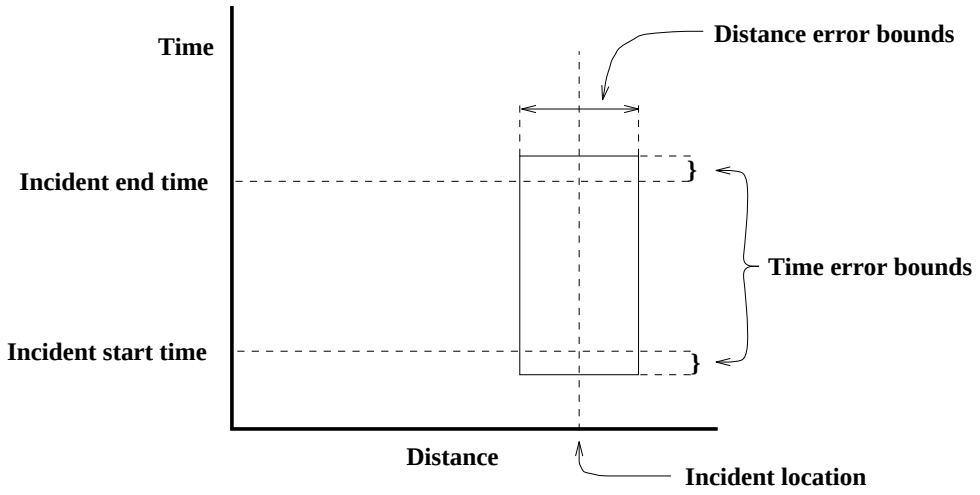

Figure 7.1: Basic Incident Plot.

YES_SPEED_TIME_PLOTS: Generate the plots.

The default for this parameter is **NO_SPEED_TIME_PLOTS**.

TIME_DIST_PLOTS Specify whether to make the time vs. distance plots of the car trajectory.

This option tells the program whether to generate the time vs. distance plots of the car data or not. If the data is filtered by specifying something for the runfile parameter CAR_SPD_FILTER_FACTOR then the filtered data is plotted. The incidents, from the key presses, are always recorded and plotted on the speed vs. time plot. If the parameter INRAD_POINTS is set to YES_INRAD_POINTS then the INRAD points are also plotted. The various options are:

NO_TIME_DIST_PLOTS: Don't generate the plots.

YES_TIME_DIST_PLOTS: Generate the plots.

The default for this parameter is **NO_TIME_DIST_PLOTS**.

TIME_ERROR_BOUND Specify the time error bound for the incident correlation analysis.

This option will tell the program to expand the size of the incident box in the time domain for the routine that does the correlation between the incident database and the car data. This parameter can be seen in Figure 7.1. This parameter expands the region of the time-distance space that the program searches for key presses to match. This option is useful if the clocks on the cars were not synchronized with each other and with the person recording the information for the incident database. Note that the time error bound is added to both sides of the box. It is obvious that if this bound is too large then the program will start to match key presses that should not be matched. For a discussion of the correlation procedure see Chapter 5.

The default for this parameter is **600**.

TRAFFIC_DELAY Specify whether to calculate the loop traffic delay or not.

This tells the program whether to calculate the loop delay or not. Note that this option only tells the program whether to physically go through and generate the loop delay files from the loop flow files. It does not tell the program whether to use a constant speed or the average speed as the threshold. The reason for this is because the calculation of the delay for each individual incident relies only on what sort of reference speed you are using. It does not care if the program has calculated the the loop delay files during this run - they could have been calculated once a long time ago. Even though this may seem like a useless option it can save you time because you won't have to calculate the loop delay files multiple times.

Note that the loop delay is not the same as the delay per incident. To calculate the loop delay the program simply reads in the loop files and calculates the delay for each loop segment for each time slice and saves this to another file. The delay per incident calculation looks at specific parts of these files to figure out the delay for a specific incident. For a complete discussion of how the loop delay is calculated see Chapter 11. The various options are:

NO_CALC_TRAFFIC_DELAY: Don't calculate the loop traffic delay.

YES_CALC_TRAFFIC_DELAY: Calculate the loop traffic delay.

The default for this parameter is **NO_CALC_TRAFFIC_DELAY**.

TRAFFIC_LOW_SPEED Specify the congestion speed.

This parameter lets the user specify what the congestion speed, in miles per hour, should be when calculating the loop delay. In order for this parameter to have any effect the runfile parameter DELAY_CALCULATION should be set to WRT_CONSTANT_SPEED. See the discussion under the parameter DELAY_CALCULATION for a short discussion of what TRAFFIC_LOW_SPEED does. For a more thorough discussion see Chapter 11.

The default for this parameter is **60**.

7.4 Default Parameters Values

Tables 7.1 - 7.5 are just a list of the default values for the various parameters. You can cause the default value to be used by either not including the parameter name in the runfile or by including the parameter name and an equals sign and then nothing else. If there is a blank space where the default should be then that means that there is no default and you have to specify a value.

7.5 Summary Of Parameter Values

Tables 7.6 - 7.14 are just a summary of all of the parameters and their options. You should probably print this little section out for future reference. If the option has an "*" then that means that you need to provide either a numerical value or a string.

Parameter	Default Value
CAR_DATA_DIRECTORY	/home/clair0/PATH/FSP/Set1/Cardata
DEBUG_LEVEL	SILENT_DEBUG
GNU_PRINTER	s307
INCIDENT_DATA_DIRECTORY	/home/clair0/PATH/FSP/Set1/Incidents
LOOP_DATA_DIRECTORY	/home/clair0/PATH/FSP/Set1/Loopdata
OUTPUT_DIRECTORY	/home/clair0/PATH/FSP/Out1

Table 7.1: Default values for the main parameters.

Parameter	Default Value
CAR_CLEANUP	DELETE_FILES
CAR_DATA_COMPRESSED	DATA_NOT_COMPRESSED
CAR_DATA_SET_NUMBER	1
CAR_DIRECTORY_ROOT	car
CAR_SPD_FILTER_FACTOR	0.9
ERROR_FILE_NAME_EXT	err
FSP_DATA_FILE_NAME	fsp.dat
GORE_POINTS	NO_CALC_GORE_POINTS
INCIDENT_POINTS	NO_INCIDENT_POINTS
INRAD_POINTS	NO_INRAD_POINTS
KEY_DATA_FILE_NAME	key.dat
MAIN_DIRECTORY	
NAV_DATA_FILE_NAME	nav.dat
REPORT_OPTION	EVERYTHING
SPEED_DIST_PLOTS	NO_SPEED_DIST_PLOTS
SPEED_TIME_PLOTS	NO_SPEED_TIME_PLOTS
TIME_DIST_PLOTS	NO_TIME_DIST_PLOTS

Table 7.2: Default values for the car parameters.

Parameter	Default Value
DELAY_CALCULATION	WRT_CONSTANT_SPEED
DELAY_TYPE	ONLY_HAVE_POSITIVE_DELAY
DROPOUT_TIMES	NO_DROPOUT_FILES
EMISSION_CALC	NO_CALC_EMISSIONS
FLOOP_CLEANUP	DELETE_EVERYTHING
GLOOP_CLEANUP	DELETE_EVERYTHING
HLOOP_CLEANUP	DELETE_NOTHING
LOOP_AGGREGATE_VALUES	NO_CALC_AGGREGATE_VALUES
LOOP_AVERAGE	NO_LOOP_AVERAGE
LOOP_CONSISTENCY_FIX	NO_FIX_CONSISTENCY_ERRORS
LOOP_DATA_COMPRESSED	DATA_NOT_COMPRESSED
LOOP_HOLES_FIX	NO_FIX_HOLE_ERRORS
LOOP_DIRECTORY	
LOOP_FILTER_FACTOR	0.9
LOOP_FLOW_PLOTS	NO_CALC_ALL_FLOW_PLOTS
LOOP_END_TIME	72000
LOOP_OUTPUT_PERIOD	60
LOOP_START_TIME	18000
LOOP_TEXT	LOOP_NO_REPORTS
OUTPUT_FLOW_AVG_FACTOR	MATCH_OUTPUT_PERIOD
PERCENT_DIESEL_TRUCKS	2.0
PERCENT_GAS_TRUCKS	3.0
TRAFFIC_DELAY	NO_CALC_TRAFFIC_DELAY
TRAFFIC_LOW_SPEED	60

Table 7.3: Default values for the loop parameters.

Parameter	Default Value
DELAY_DOWNSTREAM_NUM	0
DELAY_UPSTREAM_NUM	-1
FIX_INC_DURATION	NO_FIX_INC_DURATION
FIX_INC_LOCATION	NO_FIX_INC_LOC
HEADWAY_TIME_VAL	420
INC_DUR_EXPAND_FRACTION	50
INC_EXPLANATION	Plot
INC_FINISHED_GRAPHS	NO_FINISHED_GRAPH
INC_FINISHED_OUTPUT	NO_FINISHED_OUTPUT
INC_FINISHED_OUT_LEVEL	INC_FIN_OUT_SPARSE
INC_GRAPH_MAX_NUM	10
INC_GRAPH_MAX_PERCENT	10
INC_MATCH_ZERO_WIDTH	NO_MATCH_ZERO_WIDTH_INC
INC_RAW_MATCH_OUTPUT	NO_RAW_MATCH_OUTPUT
INC_RAW_OUTPUT_LEVEL	INC_RAW_OUT_SPARSE
PROCESS_INCIDENTS	NO_PROC_INCIDENTS

Table 7.4: Default values for the incident parameters.

Parameter	Default Value
CORRELATE_CARS_DATABASE	NO_CORRELATE
FIX_INC_DELAY_BOX	NO_FIX_INC_DELAY
INC_CONTOUR_DELAY_PLOT	NO_INC_CONTOUR_DELAY_PLOTS
INC_CORRELATION_GRAPH	NO_INC_CORR_GRAPHS
NUMBER_INC_CORR_GRAPHS	NO_NUMBER_INC_CORR_GRAPHS
TIME_ERROR_BOUND	600

Table 7.5: Default values for the analysis parameters.

Parameter	Options	Explanation
CAR_DATA_DIRECTORY	*	Car data dir
GNU_PRINTER	*	Printer name for plots
DEBUG_LEVEL		Debug stuff...
	SILENT_DEBUG	- Be very, very quite
	MINIMUM_DEBUG	- Print out sparse info
	DETAIL_DEBUG	- Print out a lot
	VERBOSE_DEBUG	- Don't use this
INCIDENT_DATA_DIRECTORY	*	Incident data dir
LOOP_DATA_DIRECTORY	*	Loop data dir
OUTPUT_DIRECTORY	*	Output dir

Table 7.6: Summary of main parameters.

Parameter	Options	Explanation
CAR_DATA_SET_NUMBER	1 or 2	Before or after study
CAR_DIRECTORY_ROOT	*	Default subdir name
CAR_SPD_FILTER_FACTOR	*	Filter factor for car speed
ERROR_FILE_NAME_EXT	*	Error file extension
FSP_DATA_FILE_NAME	*	FSP data file
KEY_DATA_FILE_NAME	*	Key data file
MAIN_DIRECTORY	*	The main sets of car data
NAV_DATA_FILE_NAME	*	Navigation data file

Table 7.7: Summary of car parameters with no pre-defined options.

Parameter	Options	Explanation
DELAY_DOWNSTREAM_NUM	*	# downstream dets. to use
DELAY_UPSTREAM_NUM	*	# upstream dets. to use
LOOP_DIRECTORY	*	The main sets of loop data
LOOP_FILTER_FACTOR	*	Filtering factor for data
PERCENT_DIESEL_TRUCKS	*	% diesel trucks
PERCENT_GAS_TRUCKS	*	% gas trucks
LOOP_END_TIME	*	End time in seconds for loop data
LOOP_OUTPUT_PERIOD	*	Seconds per period for the loop data
LOOP_START_TIME	*	Start time in seconds for loop data
TRAFFIC_LOW_SPEED	*	Traffic congestion speed

Table 7.8: Summary of loop parameters with no pre-defined options.

Parameter	Options	Explanation
HEADWAY_TIME_VAL	*	Average probe headway
INC_DUR_EXPAND_FRACTION	*	Fraction of time to take
INC_EXPLANATION	*	Title to put on plots
INC_GRAPH_MAX_NUM	*	Vertical scale on one histogram
INC_GRAPH_MAX_PERCENT	*	Vertical scale on one histogram
TIME_ERROR_BOUND	*	Width of correlation search area

Table 7.9: Summary of incident parameters with no pre-defined options.

Parameter	Options	Explanation
CAR_CLEANUP		Do what with car tmp files
	DELETE_FILES	- Delete them
	LEAVE_FILES	- Leave them there
CAR_DATA_COMPRESSED		Is car data compressed?
	DATA_NOT_COMPRESSED	- No
	DATA_IS_COMPRESSED	-Yes
GORE_POINTS		Calculate gore points?
	NO_CALC_GORE_POINTS	- No
	YES_CALC_GORE_POINTS	- Yes
INCIDENT_POINTS		Calculate incident points?
	NO_INCIDENT_POINTS	- No
	YES_INCIDENT_POINTS	- Yes
INRAD_POINTS		Calculate INRAD points?
	NO_INRAD_POINTS	- No
	YES_INRAD_POINTS	- Yes
REPORT_OPTION		Types of reports to make
	KEY_REPORT_ONLY	- Detail on the keys
	HUGE_REPORT_ONLY	- Lot's of detail
	MEDIUM_REPORT_ONLY	- More detail
	SHORT_REPORT_ONLY	- A short summary
	EVERYTHING	- Make all of the reports
SPEED_DIST_PLOTS		Make speed vs. distance plots?
	NO_SPEED_DIST_PLOTS	- No
	YES_SPEED_DIST_PLOTS	- Yes
SPEED_TIME_PLOTS		Make speed vs. time plots?
	NO_SPEED_TIME_PLOTS	- No
	YES_SPEED_TIME_PLOTS	- Yes
TIME_DIST_PLOTS		Make time vs. distance plots?
	NO_TIME_DIST_PLOTS	- No
	YES_TIME_DIST_PLOTS	- Yes

Table 7.10: Summary of pre-defined car parameters.

Parameter	Options	Explanation
DELAY_CALCULATION		How to calculate delay
	WRT_CONSTANT_SPEED	- wrt a constant speed
	WRT_AVERAGE_SPEED	- wrt an average speed
DELAY_TYPE		Can delays be negative?
	ONLY_HAVE_POSITIVE_DELAY	- No
	HAVE_POSITIVE_AND_NEGATIVE_DELAY	- Yes
DROPOUT_TIMES		Calculate the dropout times?
	NO_DROPOUT_FILES	- No
	YES_DROPOUT_FILE	- Yes
EMISSION_CALC		Calculate the emissions?
	NO_CALC_EMISSIONS	- No
	YES_CALC_CO_EMISSIONS	- Only for CO
	YES_CALC_VOC_EMISSIONS	- Only for VOC
	YES_CALC_NITRO_EMISSIONS	- Only for NITRO
	YES_CALC_ALL_EMISSIONS	- Yes - for everything
FLOOP_CLEANUP		Which temp floop files to delete
	DELETE_EVERYTHING	- All of them
	DELETE_ALL_LANES	- Just the lanes
	DELETE_MAIN_LINE_ONLY	- Just the main line
	DELETE_RAMPS_ONLY	- Just the ramps
	DELETE_NOTHING	- Leave them all
GLOOP_CLEANUP		
	DELETE_EVERYTHING	- All of them
	DELETE_MAIN_LINE_ONLY	- Just the main line
	DELETE_RAMPS_ONLY	- Just the ramps
	DELETE_NOTHING	- Leave them all
HLOOP_CLEANUP		
	DELETE_EVERYTHING	- All of them
	DELETE_MAIN_LINE_ONLY	- Just the main line
	DELETE_RAMPS_ONLY	- Just the ramps
	DELETE_NOTHING	- Leave them all

Table 7.11: Summary of pre-defined loop parameters.

Parameter	Options	Explanation
LOOP_AGGREGATE_VALUES		Calculate aggregate values?
	NO_CALC_AGGREGATE_VALUES	- No
	YES_CALC_AGGREGATE_VALUES	- Yes
LOOP_AVERAGE		Calculate the averages?
	NO_LOOP_AVERAGE	- No
	YES_LOOP_AVERAGE	- Yes
LOOP_CONSISTENCY_FIX		Fix the consistency errors?
	NO_FIX_CONSISTENCY_ERRORS	- No
	YES_FIX_CONSISTENCY_ERRORS	- Yes
LOOP_DATA_COMPRESSED		Is loop data compressed?
	DATA_NOT_COMPRESSED	- No
	DATA_IS_COMPRESSED	- Yes
LOOP_FLOW_PLOTS		Which flow plots to make
	NO_CALC_LOOP_FLOW_PLOTS	- Don't make any
	YES_CALC_OCC_FLOW_PLOTS	- Make occupancy plots
	YES_CALC_SPD_FLOW_PLOTS	- Make speed plots
	YES_CALC_PPS_FLOW_PLOTS	- Make counts plots
	YES_CALC_ALL_FLOW_PLOTS	- Make all plots
LOOP_HOLES_FIX		Fix holes in the loop data?
	NO_FIX_HOLE_ERRORS	- No
	YES_FIX_HOLE_ERRORS	- Yes
LOOP_TEXT		Whether to print data
	LOOP_NO_REPORTS	- Don't print anything
	LOOP_ERR_REPORT_ONLY	- Print error reports
	LOOP_TEXT_REPORT_ONLY	- Print text reports
	LOOP_BOTH_REPORTS	- Print both reports
OUTPUT_FLOW_AVG_FACTOR		What "PPS" means
	MATCH_OUTPUT_PERIOD	- #counts per output period
	PER_ONE_SEC	- #counts per second
TRAFFIC_DELAY		Calculate traffic delay?
	NO_CALC_TRAFFIC_DELAY	- No
	YES_CALC_TRAFFIC_DELAY	- Yes

Table 7.12: Summary of more pre-defined loop parameters.

Parameter	Options	Explanation
FIX_INC_DURATION		Fix duration errors?
	NO_FIX_INC_DURATION	- No
	FIX_INC_DURATION_FROM_DATA	- Yes, from car data
	FIX_INC_DURATION_FROM_FILE	- Yes, from made file
FIX_INC_LOCATION		Fix placement errors?
	NO_FIX_INC_LOC	- No
	YES_FIX_INC_LOC	- Yes
INC_FINISHED_GRAPHS		Generate finished graphs?
	NO_FINISHED_GRAPH	- No
	YES_FINISHED_GRAPH	- Yes
INC_FINISHED_OUTPUT		Generate finished output?
	NO_FINISHED_OUTPUT	- No
	FILE_FINISHED_OUTPUT	- Yes, to a file
	SCREEN_FINISHED_OUTPUT	- Yes, to the screen
INC_FINISHED_OUT_LEVEL		Finished output level?
	INC_FIN_OUT_SPARSE	- Don't print much
	INC_FIN_OUT_MEDIUM	- Print just enough
	INC_FIN_OUT_VERBOSE	- Print way too much
INC_MATCH_ZERO_WIDTH		Match incidents witnessed once?
	NO_MATCH_ZERO_WIDTH_INC	- No
	YES_MATCH_ZERO_WIDTH_INC	- Yes
INC_RAW_MATCH_OUTPUT		Generate raw output?
	NO_RAW_MATCH_OUTPUT	- No
	FILE_RAW_MATCH_OUTPUT	- Yes, to a file
	SCREEN_RAW_MATCH_OUTPUT	- Yes, to the screen
INC_RAW_OUTPUT_LEVEL		Raw output level?
	INC_RAW_OUT_SPARSE	- Don't print much
	INC_RAW_OUT_MEDIUM	- Print just enough
	INC_RAW_OUT_VERBOSE	- Print way too much
PROCESS_INCIDENTS		Do any processing on incidents?
	NO_PROC_INCIDENTS	- No
	YES_PROC_INCIDENTS	- Yes

Table 7.13: Summary of pre-defined incident parameters.

Parameter	Options	Explanation
CORRELATE_CARS_DATABASE		Correlate cars and incidents?
	NO_CORRELATE	- No
	YES_CORRELATE	- Yes
FIX_INC_DELAY_BOX		Use space-time boxes?
	NO_FIX_INC_DELAY	- No
	YES_FIX_INC_DELAY	- Yes
INC_CONTOUR_DELAY_PLOT		Make contour delay plots?
	NO_INC_CONTOUR_DELAY_PLOTS	- No
	YES_INC_CONTOUR_DELAY_PLOTS	- Yes
INC_CORRELATION_GRAPH		Make correlation graphs?
	NO_INC_CORR_GRAPHS	- No
	YES_INC_CORR_GRAPHS	- Yes
NUMBER_INC_CORR_GRAPHS		Put numbers on them?
	NO_NUMBER_INC_CORR_GRAPHS	- No
	YES_NUMBER_INC_CORR_GRAPHS	- Yes

Table 7.14: Summary of pre-defined analysis parameters.

Chapter 8

Runfile Parameters To xfsp Strings

Tables 8.1 and 8.2 below give a complete cross reference between the runfile parameters and the windows in the **xfsp** program. The individual options for each parameter are not listed out because they should be obvious (most of the responses are "yes" or "no"). This section has an alphabetical list of the runfile parameters and what window they reside under. Note that none of the loop tests are listed below because they all reside under the "Loop Tests" button.

8.1 xfsp Windows To Runfile Parameters

Tables 8.3 to 8.11 in this section translate from a specific **xfsp** window to a runfile parameter string.

Parameter	Window or Button	Subwindow or Title
CAR_CLEANUP	Car Output/Processing	CAR CLEANUP FILES
CAR_DATA_COMPRESSED	Car Output/Processing	CAR DATA COMPRESSED
CAR_DATA_DIRECTORY	General Options	Set Directories
CAR_DATA_SET_NUMBER	General Options	Set Directories
CAR_DIRECTORY_ROOT	Car Output/Processing	Car directory root
CAR_SPD_FILTER_FACTOR	Car Output/Processing	CAR SPEED FILTER
CORRELATE_CARS_DATABASE	Correlate Data	CORRELATE CAR OPTION
DEBUG_LEVEL	General Options	Set Main Options
DELAY_CALCULATION	Emissions/Delays	CALCULATION OPTIONS
DELAY_DOWNSTREAM_NUM	Incident Delays	Number Of Downstream Detectors
DELAY_TYPE	Emissions/Delays	DELAY TYPE OPTION
DELAY_UPSTREAM_NUM	Incident Delays	Number Of Upstream Detectors
DROPOUT_TIMES	Loop Output/Processing	DROPOUT TIMES
EMISSION_CALC	Emissions/Delays	Emission Options
ERROR_FILE_NAME_EXT	Car Output/Processing	Error file name
FIX_INC_LOCATION	Fix Inc Data	FIX INCIDENT LOCATION
FIX_INC_DELAY_BOX	Fix Inc Data	FIX INCIDENT DELAY BOX
FIX_INC_DURATION	Fix Inc Data	FIX INCIDENT DURATION
FLOOP_CLEANUP	Loop Output/Processing	LOOP CLEANUP FILES (floop)
FSP_DATA_FILE_NAME	Car Output/Processing	FSP file name
GLOOP_CLEANUP	Loop Output/Processing	LOOP CLEANUP FILES (gloop)
GNU_PRINTER	General Options	Set Main Options
GORE_POINTS	Car Output/Processing	GORE POINTS
HEADWAY_TIME_VAL	Incident Delays	HEADWAY TIME
HLOOP_CLEANUP	Loop Output/Processing	LOOP CLEANUP FILES (hloop)
INCIDENT_DATA_DIRECTORY	General Options	Set Directories
INCIDENT_POINTS	Car Output/Processing	INCIDENT POINTS
INC_CONTOUR_DELAY_PLOT	Incident Delays	CONTOUR PLOTS
INC_CORRELATION_GRAPH	Correlate Data	CORRELATION GRAPHS
INC_DUR_EXPAND_FRACTION	Fix Inc Data	Incident Duration Expand Fraction
INC_EXPLANATION	Incident Delays	GRAPH EXPLANATION
INC_FINISHED_GRAPHS	Incident Delays	FINISHED OUTPUT GRAPH
INC_FINISHED_OUTPUT	Incident Delays	FINISHED OUTPUT LOCATION
INC_FINISHED_OUT_LEVEL	Incident Delays	FINISHED OUTPUT LEVEL

Table 8.1: Runfile parameters to **xfsp** location.

Parameter	Window or Button	Subwindow or Title
INC_GRAPH_MAX_NUM	Incident Delays	Maximum number on graph
INC_GRAPH_MAX_PERCENT	Incident Delays	Maximum percentage on graph
INC_MATCH_ZERO_WIDTH	Incident Output/Processing	MATCH ZERO WIDTH
INC_RAW_MATCH_OUTPUT	Incident Output/Processing	RAW OUTPUT LOCATION
INC_RAW_OUTPUT_LEVEL	Incident Output/Processing	RAW OUTPUT LEVEL
INRAD_POINTS	Car Output/Processing	INRAD POINTS
KEY_DATA_FILE_NAME	Car Output/Processing	Key file name
LOOP_AGGREGATE_VALUES	Emissions/Delays	AGGREGATE VALUES
LOOP_AVERAGE	Loop Output/Processing	LOOP AVERAGE
LOOP_CONSISTENCY_FIX	Fix Loop Data	FIX LOOP CONSISTENCY
LOOP_DATA_COMPRESSED	Loop Output/Processing	LOOP DATA COMPRESSED
LOOP_DATA_DIRECTORY	General Options	Set Directories
LOOP_FILTER_FACTOR	Loop Output/Processing	LOOP FILTER FACTOR
LOOP_FLOW_PLOTS	Loop Output/Processing	LOOP FILES
LOOP_HOLES_FIX	Fix Loop Data	FIX LOOP HOLES
LOOP_END_TIME	Loop Output/Processing	Loop end time
LOOP_OUTPUT_PERIOD	Loop Output/Processing	Loop output period
LOOP_START_TIME	Loop Output/Processing	Loop start time
LOOP_TEXT	Loop Output/Processing	LOOP TEXT REPORTS
NAV_DATA_FILE_NAME	Car Output/Processing	Nav file name
NUMBER_INC_CORR_GRAPHS	Correlate Data	GRAPH NUMBERS
OUTPUT_DIRECTORY	General Options	Set Directories
OUTPUT_FLOW_AVG_FACTOR	Loop Output/Processing	OUTPUT FLOW FACTOR
PERCENT_DIESEL_TRUCKS	Emissions/Delays	Percent Gas Trucks
PERCENT_GAS_TRUCKS	Emissions/Delays	Percent Diesel Trucks
PROCESS_INCIDENTS	Incident Output/Processing	PROCESS INCIDENTS
REPORT_OPTION	Car Output/Processing	REPORT OPTION
SPEED_DIST_PLOTS	Car Output/Processing	SPEED DISTANCE PLOTS
SPEED_TIME_PLOTS	Car Output/Processing	SPEED TIME PLOTS
TIME_DIST_PLOTS	Car Output/Processing	TIME DISTANCE PLOTS
TIME_ERROR_BOUND	Correlate Data	TIME ERROR BOUND
TRAFFIC_DELAY	Emissions/Delays	LOOP DELAY OPTIONS
TRAFFIC_LOW_SPEED	Emissions/Delays	CONGESTION SPEED

Table 8.2: More runfile parameters to **xfsp** location.

Parameter	Window or Button	Subwindow or Title
CAR_CLEANUP	Car Output/Processing	CAR CLEANUP FILES
CAR_DATA_COMPRESSED	Car Output/Processing	CAR DATA COMPRESSED
CAR_SPD_FILTER_FACTOR	Car Output/Processing	CAR SPEED FILTER
CAR_DIRECTORY_ROOT	Car Output/Processing	Car directory root
ERROR_FILE_NAME_EXT	Car Output/Processing	Error file name
FSP_DATA_FILE_NAME	Car Output/Processing	FSP file name
GORE_POINTS	Car Output/Processing	GORE POINTS
INCIDENT_POINTS	Car Output/Processing	INCIDENT POINTS
INRAD_POINTS	Car Output/Processing	INRAD POINTS
KEY_DATA_FILE_NAME	Car Output/Processing	Key file name
NAV_DATA_FILE_NAME	Car Output/Processing	Nav file name
REPORT_OPTION	Car Output/Processing	REPORT OPTION
SPEED_DIST_PLOTS	Car Output/Processing	SPEED DISTANCE PLOTS
SPEED_TIME_PLOTS	Car Output/Processing	SPEED TIME PLOTS
TIME_DIST_PLOTS	Car Output/Processing	TIME DISTANCE PLOTS

Table 8.3: Runfile parameters in the **Car Output/Processing** window.

Parameter	Window or Button	Subwindow or Title
CORRELATE_CARS_DATABASE	Correlate Data	CORRELATE CAR OPTION
INC_CORRELATION_GRAPH	Correlate Data	CORRELATION GRAPHS
NUMBER_INC_CORR_GRAPHS	Correlate Data	GRAPH NUMBERS
TIME_ERROR_BOUND	Correlate Data	TIME ERROR BOUND

Table 8.4: Runfile parameters in the **Correlate Data** window.

Parameter	Window or Button	Subwindow or Title
DELAY_CALCULATION	Emissions/Delays	CALCULATION OPTIONS
DELAY_TYPE	Emissions/Delays	DELAY TYPE OPTION
EMISSION_CALC	Emissions/Delays	Emission Options
LOOP_AGGREGATE_VALUES	Emissions/Delays	AGGREGATE VALUES
PERCENT_DIESEL_TRUCKS	Emissions/Delays	Percent Gas Trucks
PERCENT_GAS_TRUCKS	Emissions/Delays	Percent Diesel Trucks
TRAFFIC_DELAY	Emissions/Delays	LOOP DELAY OPTIONS
TRAFFIC_LOW_SPEED	Emissions/Delays	CONGESTION SPEED

Table 8.5: Runfile parameters in the **Emissions/Delays** window.

Parameter	Window or Button	Subwindow or Title
FIX_INC_DELAY_BOX	Fix Inc Data	FIX INCIDENT DELAY BOX
FIX_INC_DURATION	Fix Inc Data	FIX INCIDENT DURATION
FIX_INC_LOCATION	Fix Inc Data	FIX INCIDENT LOCATION
INC_DUR_EXPAND_FRACTION	Fix Inc Data	Incident Duration Expand Fraction

Table 8.6: Runfile parameters in the **Fix Inc Data** window.

Parameter	Window or Button	Subwindow or Title
LOOP_CONSISTENCY_FIX	Fix Loop Data	FIX LOOP CONSISTENCY
LOOP_HOLES_FIX	Fix Loop Data	FIX LOOP HOLES

Table 8.7: Runfile parameters in the **Fix Loop Data** window.

Parameter	Window or Button	Subwindow or Title
CAR_DATA_DIRECTORY	General Options	Set Directories
CAR_DATA_SET_NUMBER	General Options	Set Directories
DEBUG_LEVEL	General Options	Set Main Options
GNU_PRINTER	General Options	Set Main Options
INCIDENT_DATA_DIRECTORY	General Options	Set Directories
LOOP_DATA_DIRECTORY	General Options	Set Directories
OUTPUT_DIRECTORY	General Options	Set Directories

Table 8.8: Runfile parameters in the **General Options** window.

Parameter	Window or Button	Subwindow or Title
DELAY_DOWNSTREAM_NUM	Incident Delays	Number Of Downstream Detectors
DELAY_UPSTREAM_NUM	Incident Delays	Number Of Upstream Detectors
HEADWAY_TIME_VAL	Incident Delays	HEADWAY TIME
INC_CONTOUR_DELAY_PLOT	Incident Delays	CONTOUR PLOTS
INC_EXPLANATION	Incident Delays	GRAPH EXPLANATION
INC_FINISHED_GRAPHS	Incident Delays	FINISHED OUTPUT GRAPH
INC_FINISHED_OUTPUT	Incident Delays	FINISHED OUTPUT LOCATION
INC_FINISHED_OUT_LEVEL	Incident Delays	FINISHED OUTPUT LEVEL
INC_GRAPH_MAX_NUM	Incident Delays	Maximum number on graph
INC_GRAPH_MAX_PERCENT	Incident Delays	Maximum percentage on graph

Table 8.9: Runfile parameters in the **Incident Delays** window.

Parameter	Window or Button	Subwindow or Title
INC_MATCH_ZERO_WIDTH	Incident Output/Processing	MATCH ZERO WIDTH
INC_RAW_OUTPUT_LEVEL	Incident Output/Processing	RAW OUTPUT LEVEL
INC_RAW_MATCH_OUTPUT	Incident Output/Processing	RAW OUTPUT LOCATION
PROCESS_INCIDENTS	Incident Output/Processing	PROCESS INCIDENTS

Table 8.10: Runfile parameters in the **Incident Output/Processing** window.

Parameter	Window or Button	Subwindow or Title
DROPOUT_TIMES	Loop Output/Processing	DROPOUT TIMES
FLOOP_CLEANUP	Loop Output/Processing	LOOP CLEANUP FILES (floop)
GLOOP_CLEANUP	Loop Output/Processing	LOOP CLEANUP FILES (gloop)
HLOOP_CLEANUP	Loop Output/Processing	LOOP CLEANUP FILES (hloop)
LOOP_AVERAGE	Loop Output/Processing	LOOP AVERAGE
LOOP_DATA_COMPRESSED	Loop Output/Processing	LOOP DATA COMPRESSED
LOOP_FILTER_FACTOR	Loop Output/Processing	LOOP FILTER FACTOR
LOOP_FLOW_PLOTS	Loop Output/Processing	LOOP FILES
LOOP_END_TIME	Loop Output/Processing	Loop end time
LOOP_OUTPUT_PERIOD	Loop Output/Processing	Loop output period
LOOP_START_TIME	Loop Output/Processing	Loop start time
LOOP_TEXT	Loop Output/Processing	LOOP TEXT REPORTS
OUTPUT_FLOW_AVG_FACTOR	Loop Output/Processing	OUTPUT FLOW FACTOR

Table 8.11: Runfile parameters in the **Loop Output/Processing** window.

Chapter 9

Program Input: The Incident Filter

The second parameter that the **fsp** program takes on the command line is the name of an incident filter. The incident filter tells the program which incidents to pull out of the incident database. All of the processing on the incidents is done on the filtered set of incidents. This chapter explains how to use the incident filter and its required format.

9.1 The Incident Filter Format

The format of the incident filter is pretty straight forward. It is a text file that can be edited with any text editor. Each line indicates one field in the incident database that you wish to filter. Every line has a descriptor word followed by an equals sign followed by the desired values. For a list of all of the descriptor words see Section 9.2. An example of a line in an incident filter is given below:

```
SHIFT                   = 0
```

In the line above the descriptor word is "SHIFT" and the value that we are looking for is "0." If this line was the only line in your incident filter then you would be telling the program to only allow the incidents whose shift field was 0 through the filter. When I say that an incident is allowed through the filter it means that the program will read that incident from the incident database and do more processing on it. If there are more lines in the incident filter then the final filter is a logical "AND" of all of the lines. So if the two lines in the filter were:

```
SHIFT                   = 0
DIRECTION               = 1
```

The the filter would only match incidents that had a shift of 0 AND a direction of 1. Probably the most important conceptual thing to understand about the incident filter is that it can not do a logical "OR" across fields. You can not have the incident filter match incidents that had a shift of 0 or had a direction of 1. But you can do a logical "OR" within a single field. If there is a field with a few different possible values, like the automobile color field which has values in the range 0-12, then the incident filter can match multiple values. You specify this by typing each value that you want the filter to match in that field on the same line:

```
VEHICLE_1_COLOR         = 1 4 7
```

This would match any vehicle that had a color of black, gold, or orange. If there are two lines like this in the incident filter then the filter still performs an "AND" across the fields and an "OR" within the fields. So if you had something like:

```
VEHICLE_1_COLOR         = 1 4 7
VEHICLE_1_TYPE          = 2 3 4
```

Then this would look for incidents that were black, gold, or orange and that were a pickup truck, van, or station wagon. Another thing to note about the incident filter is that it can't do a logical "NOT" of a field. So if you wanted to look for all incidents where the vehicles were not red then you would have to tell the filter to specifically look for all of the other colors besides red.

While most of the incident fields are simply integer values a few of them are time or date values. These are specified in the incident filter in a pretty straight forward way as seen by the examples below:

```
DATE                    = 2/16/93 - 2/18/93
TIME                    = 16:00 - 19:00
```

The date field above is listed out from the starting date to the ending date in the standard *month/day/year* format. The date field is closed on the left and open on the right. This means that the left date is included but that the right date is not. To specify one day then make the dates one day apart. Note that there needs to be a space between the "-" and the values. The time field has the starting and ending time values in military time. If you mess up the times and list out an ending time that is before the starting time then the incident filter will not match anything. So the incident filter above will match incidents that occurred on either February 16th or 17th and happened between 4pm and 7pm.

Finally, the "#" character is a comment character. So if a line starts with the "#" character then the whole line is discarded. This is useful if you want to rapidly switch between options in an incident filter without retyping.

9.2 Fields Of The Incident Filter

As was mentioned above, when the **fsp** program reads in the incident filter it expects there to be a one word field descriptor at the start of each line. In Tables 9.1 and 9.2 I have listed out the field descriptor words that you should use for each field.

9.3 Incident Filter Examples

Below are a few examples of how to use the incident filter to obtain some interesting results. Since the runfile dictates how the incidents are processed I have listed out the relevant runfile parameters as well. Note that the numbers in parentheses to the right of each line are not part of the incident filter or the runfile - they are only there for reference purposes. There are a few

Column	*Name*	*Field Descriptor*	*Field Type*
A	Type	DATA_TYPE	char (F,C,T)
B	Incident	INC_NUMBER	integer
C	Date	DATE	date
D	Shift	SHIFT	integer
E	Time	TIME	time
F	Direction	DIRECTION	integer
G	Beginning	AT_BEGINNING	integer
H	End	AT_END	integer
I	Link Identity	LINK_ID	integer
J	Location	LOCATION	integer
K	Relative	RELATIVE_LOC	integer
L	Exit Distance	EXIT_DISTANCE	integer
M	Primary Lane	LANE_1_AFFECTED	integer
N	2nd Lane	LANE_2_AFFECTED	integer
O	3rd Lane	LANE_3_AFFECTED	integer
P	Incident Type	INCIDENT_DESCRIPTION	integer
Q	Type 1	INCIDENT_TYPE_1	integer
R	Type 2	INCIDENT_TYPE_2	integer
T	Begin/End	BEGIN_END	integer
S	Type 3	INCIDENT_TYPE_3	integer
U	Num Vehicles	NUM_VEHICLES	integer
V	Vehicle 1 Type	VEHICLE_1_TYPE	integer
W	Vehicle 2 Type	VEHICLE_2_TYPE	integer
X	Vehicle 3 Type	VEHICLE_3_TYPE	integer
Y	Vehicle 1 Color	VEHICLE_1_COLOR	integer
Z	Vehicle 2 Color	VEHICLE_2_COLOR	integer

Table 9.1: Some field descriptors for incident filter.

steps that you want to follow every time that you want to create an incident filter and process the incidents. Below I have listed out two different lists: a short list and a long list. The short list gives a very quick overview of the general principles of filtering incidents. The longer list spells out most of the steps that you need to follow in order to do any filtering. The short list:

1. Specify what incidents you want to filter.
2. Specify what kind of fixes you want to perform on the incidents.
3. Specify any loop or car data that you need for the fixes you chose.
4. Specify what kind of output you want.

These four steps encompass all that you need to do to filter and process incidents. These steps are expanded out in more detail below in the longer list:

1. Decide which incidents you want to filter:

Column	*Name*	*Field Descriptor*	*Field Type*
AA	Vehicle 3 Color	VEHICLE_3_COLOR	integer
AB	Ticket For Tow	TICKETED	integer
AC	CHP	CHP	integer
AD	Entries In Log	NUM_TIMES_IN_LOG	integer
BF	No Tow	TT_NO_REMOVE	integer
BG	Main Clear	TIME_MAIN_CLEAR	time
BH	FSP Arrival	FSP_ARRIVAL	integer
BI	CHP Arrival	CHP_ARRIVAL	time
BJ	Tow Truck Arrival	TOW_ARRIVAL	time
BK	Ambulance Arrival	AMB_ARRIVAL	time
BL	Fire Arrival	FIRE_ARRIVAL	time
BM	CHP Departure	CHP_DEPART	time
BN	Tow Truck Departure	TOW_DEPART	time
BO	Ambulance Departure	AMB_DEPART	time
BP	Fire Departure	FIRE_DEPART	time
BQ	Comments	COMMENTS	integer
BR	Official	NUM_OFFICIAL	integer
BS	Non-Official	NUM_NON_OFFICIAL	integer
BT	Two Truck Response	TOW_TRUCK_RESPONSE	time
BU	Two Truck Clearance	TOW_TRUCK_CLEARANCE	time
CX	Duration	DURATION	time
CY	Weather	WEATHER	integer

Table 9.2: More field descriptors for incident filter.

   (a) Figure out what the field descriptors are for the fields that you want.

   (b) Figure out what the values are for each field that you are interested in.

   (c) Create the incident filter.

2. Set the runfile parameter PROCESS_INCIDENTS to YES_PROC_INCIDENTS.

3. Decide if you want to match incidents witnessed once and set the runfile parameter INC_MATCH_ZERO_WIDTH.

4. Decide what kind of fixes you want to perform on the incident data:

   (a) Decide if you want to fix the duration of the incidents and set the runfile parameters FIX_INC_DURATION and INC_DUR_EXPAND_FRACTION.

   (b) Decide if you want to fix the location of the incidents and set the runfile parameters FIX_INC_LOCATION and CORRELATE_CARS_DATABASE.

   (c) Decide if you want to fix the bounding box of the incidents for the delay calculation and set the runfile parameter FIX_INC_DELAY_BOX.

5. Decide what kind of output you want on the raw incidents and set the runfile parameters INC_RAW_MATCH_OUTPUT and INC_RAW_OUTPUT_LEVEL.

6. Decide what kind of output you want on the finished incidents and set the runfile parameters INC_FINISHED_OUTPUT and INC_FINISHED_OUT_LEVEL.

7. Decide if you want any graphs:

   (a) Decide if you want to generate a graph of the delay vs. duration and set the runfile parameter INC_FINISHED_GRAPHS.

   (b) Decide what you want the title to be and set the runfile parameter INC_EXPLANATION.

   (c) Decide what you want the scales on the graphs to be and set the runfile parameters INC_GRAPH_MAX_NUM and INC_GRAPH_MAX_PERCENT.

8. Decide if you want (or need) to process any other loop or car data and set those parameters accordingly.

9. Set the general parameters like the debug level and data directories.

9.3.1 Example 1: Examining Incident Fields

For the first example we are not going to do any processing on the incidents at all. We are just going to list out all of the incident fields. The way that we will do this is by specifying the incident number that we want to examine in the incident filter like so:

```
INC_NUMBER              = 16                (1)
```

This will tell the incident filter to only pull out incidents that have an incident number of 16. Of course, there should only be one incident in the incident database that has an incident number of 16. If we wanted to pull out multiple incidents, like incidents 16-20, we could have an incident filter that looks like this:

```
INC_NUMBER              = 16 17 18 19 20    (1)
```

Note that there aren't any commas between the various numbers. If you wanted the incident filter to extract a lot of incidents then the incident numbers all have to be on the same line - you can't specify two lines for the same field. The only restriction is that any single line can't be more than 2000 characters long. Hopefully, this is long enough for most needs. To just list out the fields of these incidents the following runfile would work just fine:

```
PROCESS_INCIDENTS       = YES_PROC_INCIDENTS            (1)
INC_RAW_MATCH_OUTPUT    = SCREEN_RAW_MATCH_OUTPUT       (2)
INC_RAW_OUTPUT_LEVEL    = INC_RAW_OUT_VERBOSE           (3)
INC_FINISHED_OUTPUT     = NO_FINISHED_OUTPUT            (4)
```

Line by line, these runfile parameters will do the following:

1) Process the incident data.

2) Send the raw incident output to the screen.

3) Generate a lot of output for the raw incident output - this will print out all of the incident fields for every incident that matched the filter. There are approximately 50 fields per incident that are printed out.

4) Don't generate any finished output on the incidents.

There are a couple of things that were not listed out in the runfile above. We didn't list out any general parameters like DEBUG_LEVEL or GNU_PRINTER, we didn't specify any parameters that dealt with loop or car data, and we didn't list out any parameters that dealt fixing the incident data. We didn't specify these parameters because we are assuming that if a parameter is not listed in the runfile then it is set to NO or OFF - whichever is appropriate. For a more complete discussion of the various types of incident output see Chapter 16.

9.3.2 Example 2: Accidents With Little Processing

For this example let's filter out all of the incidents that are in-lane accidents. The incident filter should look like this:

```
DATA_TYPE              = F              (1)
LANE_1_AFFECTED        = 1 2 3 4 5      (2)
INCIDENT_TYPE_2        = 1 2            (3)
```

This will pull out all of the incidents that are (1) field data, (2) that occur in lanes 1-5, and (3) that are either a single car or multi-car accident. We assume here that if the accident occurs in one of the lanes that that lane will be listed in the filter field LANE_1_AFFECTED. If we just wanted to pull out these incidents and list them out without finding the delay or doing any fixing of the incident data then we could have a runfile that looks like the following:

```
PROCESS_INCIDENTS       = YES_PROC_INCIDENTS            (1)
INC_MATCH_ZERO_WIDTH    = YES_MATCH_ZERO_WIDTH_INC      (2)
INC_RAW_MATCH_OUTPUT    = SCREEN_RAW_MATCH_OUTPUT       (3)
INC_RAW_OUTPUT_LEVEL    = INC_RAW_OUT_SPARSE            (4)
INC_FINISHED_OUTPUT     = SCREEN_FINISHED_OUTPUT        (5)
INC_FINISHED_OUT_LEVEL  = INC_FIN_OUT_MEDIUM            (6)
HEADWAY_TIME_VAL        = 0                             (7)
```

Note that this runfile is not complete. The parameters listed out are only the ones specific to processing the incident data. The lines in the runfile do the following (listed according to line number):

1) Process the incident data.

2) Include incidents that were only witnessed once.

3) Send any preprocessed incident output to the screen.

4) Don't generate too much preprocessed incident output.

5) Send any processed incident output to the screen.

6) Generate a medium amount of processed incident output.

7) Set the additive headway value to zero.

This combination of incident filter and runfile will do the least amount of processing and the produce the least amount of output. There will be no attempt to fix the incident location, duration, or to get the proper delay.

9.3.3 Example 3: Red Cars With Lots Of Processing

For this example let's look at the incidents that happened in the morning in the southbound direction between A-Street and Whipple that involved a red car but that was not a ticketing incident. This incident filter would look like this:

```
DATA_TYPE              = F                (1)
SHIFT                  = 0                (2)
DIRECTION              = 1                (3)
LOCATION               = 4 5 6 7 8 9      (4)
VEHICLE_1_COLOR        = 8                (5)
INCIDENT_TYPE_3        = 0                (6)
```

And if we wanted to do a bit more processing with these incidents our runfile might look something like this:

```
PROCESS_INCIDENTS       = YES_PROC_INCIDENTS              (1)
FIX_INCIDENT_DATA       = YES_FIX_INC_DATA                (2)
CORRELATE_CARS_DATABASE = NO_CORRELATE                    (3)
FIX_INC_DURATION        = FIX_INC_DURATION_FROM_FILE      (4)
INC_DUR_EXPAND_FRACTION = 50                              (5)
FIX_INC_DELAY_BOX       = YES_FIX_INC_DELAY               (6)
INC_CONTOUR_DELAY_PLOT  = YES_INC_CONTOUR_DELAY_PLOTS     (7)
INC_MATCH_ZERO_WIDTH    = YES_MATCH_ZERO_WIDTH_INC        (8)
INC_RAW_MATCH_OUTPUT    = SCREEN_RAW_MATCH_OUTPUT         (9)
INC_RAW_OUTPUT_LEVEL    = INC_RAW_OUT_SPARSE              (10)
INC_FINISHED_OUTPUT     = SCREEN_FINISHED_OUTPUT          (11)
INC_FINISHED_OUT_LEVEL  = INC_FIN_OUT_MEDIUM              (12)
INC_FINISHED_GRAPHS     = YES_FINISHED_GRAPH              (13)
INC_EXPLANATION         = Morning, southbound, red cars   (14)
INC_GRAPH_MAX_NUM       = 10                              (15)
INC_GRAPH_MAX_PERCENT   = 20                              (16)
HEADWAY_TIME_VAL        = 0                               (17)
```

Where the lines in the runfile do the following (listed according to line number):

1) Process the incident data.

2) Attempt to fix the location of the incidents from a file read in at runtime.

3) Don't attempt to correlate the incident database with the car data. This is taken care of by item 2.
4) Attempt to fix the duration of the incidents from a file read in at runtime.
5) Expand the starting and ending times of the incident 50% of the way to when a different car passed this location without spotting the incident.
6) Attempt to set the bounding box for the delay of the incidents from a file read in at runtime.
7) Generate the contour, density, and differential density plots of the loop detector data overlaid with the incident locations.
8) Include incidents that were only witnessed once.
9) Send any preprocessed incident output to the screen.
10) Don't generate too much preprocessed incident output.
11) Send any processed incident output to the screen.
12) Generate a medium amount of processed incident output.
13) Generate a graph of the delay vs duration for all of the incidents.
14) Put the title "Morning, southbound, red cars" on the plots.
15) Set the vertical scale to be 10 cars on one histogram plot.
16) Set the vertical scale to be 20 percent on another histogram plot.
17) Set the additive headway value to zero since this is taken care of by items 4 and 5.

9.3.4 Example 4: Tow Truck Incidents

In this example we are trying to list out all of the incidents that occurred on March 16th or 17th that had a tow truck respond. These are referred to as the assisted incidents. We also want to exclude incidents that started before we arrived or incidents that ended after we left.

```
DATA_TYPE              = F                  (1)
DATE                   = 2/16/93 - 2/18/93  (2)
TOW_ARRIVAL            = 6:00 - 20:00       (3)
BEGIN_END              = 0                  (4)
```

We list out the tow truck arrival time in line 3 because in order to know if a tow truck showed up we can just check to see if a tow truck arrival time is listed. If it is then we are assured that a tow truck arrived. The BEGIN_END field in line 4 was a field that we added that is the logical "OR" of fields G and H. A value of "0" for this field means that we are excluding incidents that started before we got there and incidents that ended after we left. So the only incidents that we want to look at are ones that started and ended during our shift. In this example we are going to assume that we want to process everything from scratch. The runfile is listed out below in various parts according to function. The first section of the runfile deals with the car data:

```
MAIN_DIRECTORY      = am021693 1 2 3 4          (1)
```

```
MAIN_DIRECTORY      = pm021693 2 3 4            (1)
MAIN_DIRECTORY      = am021793 1 3 4            (1)
MAIN_DIRECTORY      = pm021793 1 2 3 4          (1)
GORE_POINT_OPTION   = YES_CALC_GORE_POINTS      (2)
INCIDENT_POINTS     = YES_INCIDENT_POINTS       (3)
INRAD_POINTS        = YES_INRAD_POINTS          (4)
SPEED_TIME_PLOTS    = YES_SPEED_TIME_PLOTS      (5)
SPEED_DIST_PLOTS    = YES_SPEED_DIST_PLOTS      (6)
TIME_DIST_PLOTS     = YES_TIME_DIST_PLOTS       (7)
```

This section of the runfile will process the car data so that it can be used later on when the program needs to fix the incident durations and locations. Line by line, this section does the following:

1) Simply list out the car data that we have available for the days of February 16th, and 17th.
2) Find all of the gore points in the car data - used when attempting to fix the incident locations.
3) Find all of the places were the drivers pressed an incident key - used when attempting to fix the incident locations.
4) Find all of the INRAD points in the car data - once again, used when attempting to fix the incident locations.
5) Generate the speed vs. time plots - just nice to have.
6) Generate the speed vs. distance plots - just nice to have.
7) Generate the distance vs. time plots - needed to find the incident durations.

As you can see, the part of the runfile that deals with the car data basically tells the program to process the car data and to record everything. The next section of the runfile deals with the loop data that we need:

```
LOOP_DIRECTORY = lp021693 1 2 3 4 5 6 7 8 9 10 11 12 13 15 16 17 19 20  (1)
LOOP_DIRECTORY = lp021793 1 2 3 4 5 7 8 9 10 11 12 13 15 16 17 19 20  (1)
LOOP_START_TIME           = 18000                                      (2)
LOOP_END_TIME             = 72000                                      (3)
LOOP_OUTPUT_PERIOD        = 300                                        (4)
OUTPUT_FLOW_AVG_FACTOR    = MATCH_OUTPUT_PERIOD                        (5)
LOOP_FLOW_PLOTS           = YES_CALC_ALL_FLOW_PLOTS                    (6)
LOOP_TEXT                 = LOOP_ERR_REPORT_ONLY                       (7)
DROPOUT_TIMES             = YES_DROPOUT_FILE                           (8)
LOOP_HOLES_FIX            = YES_FIX_HOLE_ERRORS                        (9)
TRAFFIC_DELAY             = YES_CALC_TRAFFIC_DELAY                     (10)
DELAY_CALCULATION         = WRT_CONSTANT_SPEED                         (11)
DELAY_TYPE                = HAVE_POSITIVE_AND_NEGATIVE_DELAY           (12)
TRAFFIC_LOW_SPEED         = 55                                         (13)
```

Much like the last section of the runfile, this section will massage the loop data such that it can be used later on when processing the incident data. Line by line, this section of the runfile does the following:

1) Simply list out the loop data that we have available for the days of February 16th and 17th.
2) Make the starting time for the loop data 18000 seconds since midnight - 5am. This is the starting time of our loop data.
3) Make the ending time for the loop data 72000 seconds since midnight - 8pm. This is the ending time of our loop data.
4) Generate the loop data every 300 seconds.
5) Make the PPS file be "Counts per output period." See the discussion in Chapter 7 under the parameter OUTPUT_FLOW_AVG_FACTOR for more information.
6) Generate the flow, occupancy, and speed files from the loop data.
7) Generate an error report for the loop data - this is needed to fix the holes in the loop data.
8) Generate a report on the holes in the loop data - this is needed to fix the holes in the loop data.
9) Actually fix the holes in the loop data. This will take the floop files and generate the gloop files.
10) Calculate the traffic delay for each loop. This is used when calculating the delay per incident.
11) When calculating the delay at each loop, perform the calculation with respect to a constant congestion speed.
12) When calculating the delay at each loop, allow the delay to become negative.
13) Use 55 mph as the congestion speed in the delay calculation.

Finally, this last section of the runfile deals with the processing of the incidents themselves:

```
PROCESS_INCIDENTS           = YES_PROC_INCIDENTS           (1)
CORRELATE_CARS_DATABASE     = YES_CORRELATE                (2)
TIME_ERROR_BOUND            = 360                          (3)
INC_CORRELATION_GRAPH       = YES_INC_CORR_GRAPHS          (4)
NUMBER_INC_CORR_GRAPHS      = YES_NUMBER_INC_CORR_GRAPHS   (5)
FIX_INC_DURATION            = FIX_INC_DURATION_FROM_DATA   (6)
INC_DUR_EXPAND_FRACTION     = 100                          (7)
INC_CONTOUR_DELAY_PLOT      = YES_INC_CONTOUR_DELAY_PLOTS  (8)
INC_RAW_MATCH_OUTPUT        = SCREEN_RAW_MATCH_OUTPUT      (9)
INC_RAW_OUTPUT_LEVEL        = INC_RAW_OUT_SPARSE           (10)
INC_FINISHED_OUTPUT         = SCREEN_FINISHED_OUTPUT       (11)
INC_FINISHED_OUT_LEVEL      = INC_FIN_OUT_MEDIUM           (12)
INC_FINISHED_GRAPHS         = YES_FINISHED_GRAPH           (13)
```

```
INC_EXPLANATION             = February 16th and 17th      (14)
INC_GRAPH_MAX_NUM           = 10                          (15)
INC_GRAPH_MAX_PERCENT       = 20                          (16)
HEADWAY_TIME_VAL            = 0                           (17)
```

This last section tells the program to use the car and loop data that was processed earlier to attempt to fix the durations and locations of the incidents. Line by line, this section of the runfile does the following:

1) Simply tells the program to process the incidents.
2) Attempt to correlate the incidents that made it through the filter with the car data that was processed. This will fix the locations of the incidents.
3) When doing the correlation expand the incident duration by 360 seconds. See the discussion in Section 5.3.1 for a complete explanation.
4) Make the correlation graphs that give a visual representation of how the car data matches up with the incident database.
5) Put the incident numbers on the correlation graphs.
6) Attempt to fix the incident durations from the car data.
7) When fixing the durations expand the incident start or end time 100% of the way to when the last car went by the incident location.
8) Make the loop contour plots of delay, density and differential density with the incidents on them.
9) Send any preprocessed incident output to the screen.
10) Don't send too much preprocessed incident output anywhere.
11) Send any processed incident output to the screen.
12) Generate a medium amount of processed incident output.
13) Generate a graph of the delay vs duration for all of the incidents.
14) Put the title "February 16th and 17th" on the plots.
15) Set the vertical scale to be 10 cars on the histogram plot of the number of incidents.
16) Set the vertical scale to be 20 percent on the histogram plot of the percentage of incidents.
17) Set the additive headway value to zero since this is taken care of by items 6 and 7.

The results of this search are given below:

```
Individual incident statistics:
              Inc. Type                              Good   Bad
Inc #   Date   D 1 2 3   Time South Link Loop  Files Files  Duration     Delay
   4  2/16/93  0 0 2 0   7:34   1    13    12    13     0     0:26:01      7.53
  43  2/16/93  0 0 2 0  16:11   0     7    19     7     0     0:39:44     37.81
  50  2/16/93  0 5 0 0  17:40   0     1     5     1     0     0:30:24      0.53
  68  2/17/93  0 3 0 0   8:38   1    16    15    16     0     0:17:41     28.27
```

```
  87  2/17/93  0 5 0 0  17:23   1    5    20     5     0     1:18:17     10.96

Stats on all incident delays:
   Match incidents witnessed once = YES
   ALL results include headway time of = 0:00
   Number of Incidents   = 5
   Number witnessed once = 1
                        Min     Max     Mean     Std. Dev.    Std. Error
   Incident Duration     17      78     38.42      23.66        10.58
   TT Response  (  5)     0      31      7.60      13.43         6.00
   TT Clearance (  5)     0      28      7.40      11.78         5.27
   Incident Delay      0.53   37.81     17.02      15.47         6.92
```

Chapter 10

Program Input: The Loop Detector Tests

At the start of the project there was great concern as to whether the loop detectors were performing properly. To address this concern there were a number of tests that were devised to detect problems with the loop data. This chapter describes all of these tests, what parameters they take, and how to initiate them.

10.1 Generating The Tests

Each test has a couple of runfile parameters associated with it. One of the parameters is a simple on or off flag to indicate whether to run this test at all. The other parameters for a test are needed to specify the conditions of the test. For example, there is a test to flag when the speed goes above a certain value. In order to run this test you need to set the main parameter, **LP_SPEED_HIGH_TEST**, to **YES**. An example of this in a runfile is:

```
LP_SPEED_HIGH_TEST = YES
```

Once that is done you need to specify what speed you want to have flagged. This is done with the parameter **LP_SPEED_HIGH_THRESHOLD_MPH** as in the example below:

```
LP_SPEED_HIGH_TEST = YES
LP_SPEED_HIGH_THRESHOLD_MPH = 65
```

Finally, since the loop data can tend to be noisy, you can specify the number of times that the speed has to be above the threshold value before it is flagged. Remember that the speed on each lane is extracted once every time period as specified by the parameter **LOOP_OUTPUT_PERIOD**. So if you have the output value set to 60 seconds and you want to flag every time that the speed goes above 65 mph for three times in a row then that would mean that the speed would need to be above 65 mph for three minutes before it is flagged. The way to do this is with the parameter **LP_SPEED_HIGH_THRESHOLD_NUM**:

```
LP_SPEED_HIGH_TEST = YES
LP_SPEED_HIGH_THRESHOLD_MPH = 65
LP_SPEED_HIGH_THRESHOLD_NUM = 3
```

If the speed should go above 65 mph for only one or two samples then it will not be flagged at all. A graphical illustration of this is given in Figure 10.1.

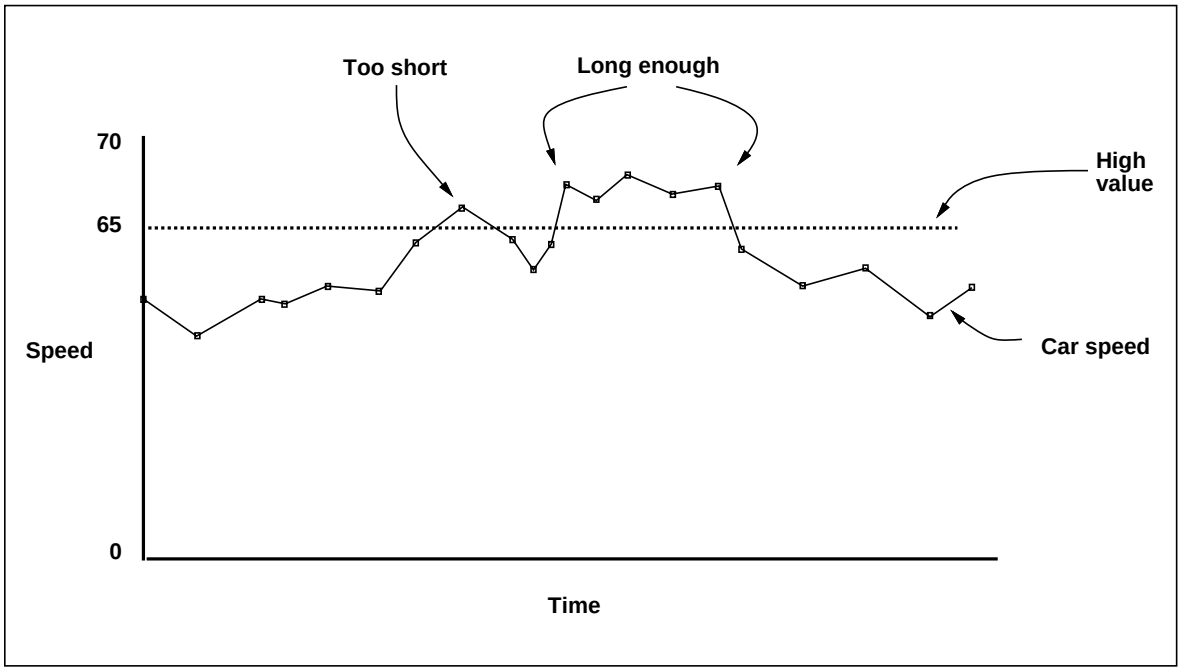

Figure 10.1: High Speed Test.

Figure 10.1 is a drawing of a fictional speed time plot for some lane on the freeway. The small squares are the actual data points from the loop data and the line just connects them. In reality all of these points would be evenly spaced. In this figure, the speed goes above the high value twice. The first time that it does so it only stays above the high value for one time period, therefore that event is not flagged. The second time that the speed goes above the high threshold it stays there for five time periods. Since this is longer than the value specified by **LP_SPEED_HIGH_THRESHOLD_NUM** this would be flagged.

When I say that an event will be flagged then that means that an entry will be made in the error file. An entry contains the starting time of the event, the ending time, the duration (for those of us who can't subtract), the trap number, and a short description of the problem. An example of an entry in an error file for our example above is:

```
  From     Till    Duration    Trap #      Problem
6:00:00   6:04:00   0:04:00    1          SPEED passed high threshold.
```

This says that the problem started at 6:00am and continued until 6:04am; a total of 4 minutes. It was at trap 1 and the problem was that the speed passed the high threshold that we set.

There are a few other types of tests. They are "cross" tests and "range" tests. In a cross test we are checking that the values between lanes don't exceed a certain percentage: we are checking across lanes. One of these tests is the cross occupancy test. This is specified

by the test parameter **LP_CROSS_OCC_TEST**. If we wanted to check that the values of the occupancies didn't change more than 5 percent from lane to lane then we would specify the following parameters:

```
LP_CROSS_OCC_TEST               = YES
LP_CROSS_OCC_RANGE_PERCENT      = 5
```

A graphical illustration of this is given in Figure 10.2.

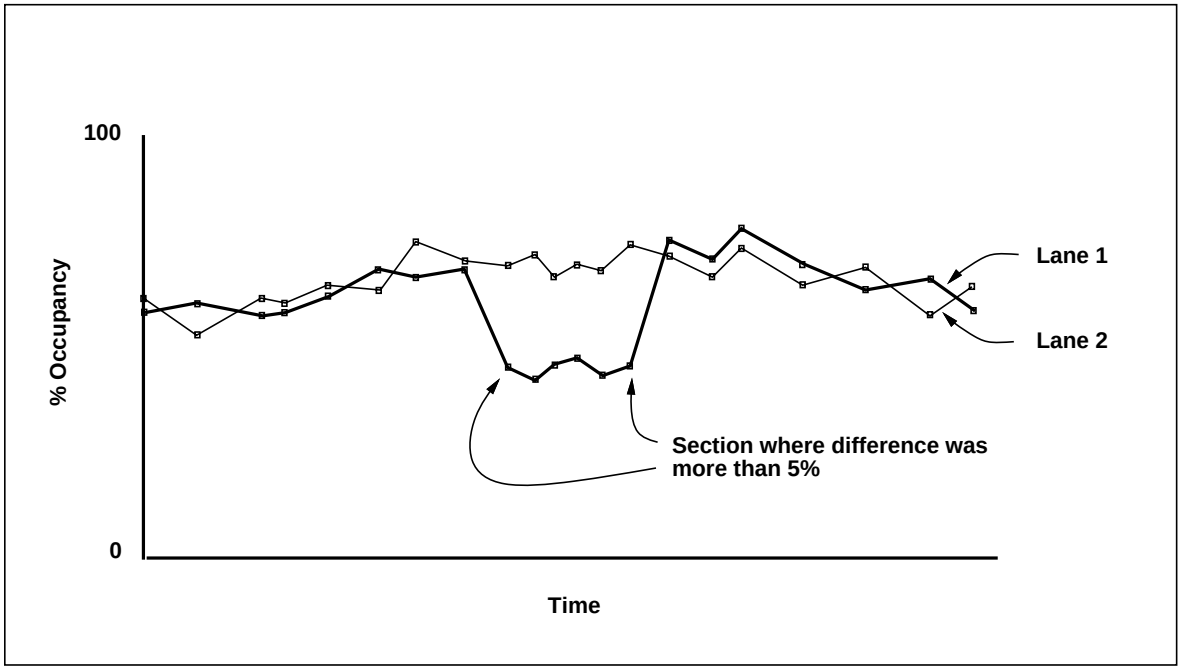

Figure 10.2: Cross Lane Test.

Figure 10.3 is a picture of what the typical section of a loop detector looks like. There are four lanes and the boxes represent the loop detectors. In the cross tests we test the values between the lanes as indicated on the figure. The program knows what the layout of the freeway is and it knows to only check adjacent lanes. On this particular section of freeway for the cross lane occupancy test, the program would check that the occupancies match within 5 percent of each other for lanes 1 and 2, lanes 2 and 3, and lanes 3 and 4, for both the upstream and downstream detectors. There are four different cross tests that you can run: occupancies, on times, speeds, and counts. The runfile parameters are:

```
LP_CROSS_OCC_TEST
LP_CROSS_ON_TEST
LP_CROSS_PPS_TEST
LP_CROSS_SPEED_TEST
```

The range test is basically the same as the cross test except that you test between the upstream and downstream detectors in the same pair. There are two different range test that you can run: occupancies, and counts. The runfile parameters are:

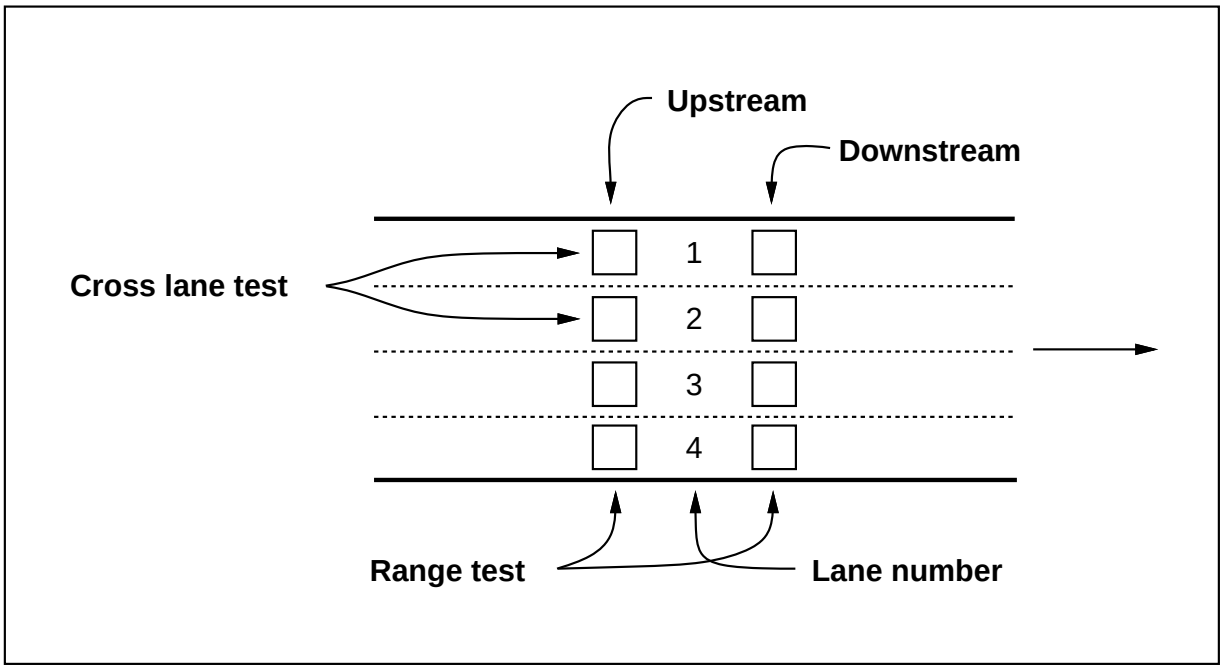

Figure 10.3: The Loop Detectors In The Freeway.

```
LP_OCC_RANGE_TEST
LP_PPS_RANGE_TEST
```

There are some things to point out about the error reports.

- The error criterion that you specify are run on every trap, on every detector, and for every day that you specify. If you specify a lot ot tests to run then this will slow down the analysis quite a bit, so be prepared to wait.
- For every parameter there are defaults. The defaults are listed out below in the Table 10.2.
- To turn off a specify test then just set the main parameter to **NO** instead of **YES**.
- At the top of every error file is a listing of the criterion for every test. This is helpful in remembering what tests were actually run.
- All of the error reports for a certain day are placed in one file in the directory defined in the include file `fsp.h` by the defined variable **LOOPDATA_SUMMARY_DIR**. The error file name corresponds to the name of the data directory with the extension **.sum**. So if you were generating the error report for all of the loop data in the directory `lp030193` then the error reports for all of the days is placed in the file `lp030193.sum` in the directory **LOOPDATA_SUMMARY_DIR**. Currently, this directory is defined as "Reports" and it is always located under the loop output directory.
- If there are tests being done on something other than speed then there is an extra piece of information in the error file entry. This is the specific trap which encountered the problem.

It could have been the upstream trap or the downstream trap. The speed doesn't have this feature because it takes two traps to generate speed data, so there is no upstream or downstream. The extra information looks like this:

```
  From     Till    Duration    Trap #      Problem
6:30:00  6:54:00   0:24:00    1 - Down    OCC passed range threshold.
```

Where **Down** means downstream and **Up**, paradoxically enough, means upstream.

- If the parameter **DROPOUT_TIMES** is specified correctly then there will be information at the end of the error report on the times that the loop detector didn't collect data.

10.2 Listing Of The Various Tests

All of the tests that can be run on the loop data are listed out below. The parameter to turn the test on or off is referred to as the main parameter and the other parameters are called the auxiliary parameters. The various tests are listed out below alphabetically according to the main parameter. The value given for the typical entry is the default value for the various tests.

LP_CROSS_OCC_TEST This allows the user to specify the maximum percentage change in occupancy between lanes that is allowable.

OUTPUT IN ERROR FILE: The error file entry contains the two offending lanes in a two character code. The first character is either **S**, for southbound, or **N**, for northbound, and the second character is the lane number. In this example the occupancy in the southbound lane 4 was more than the specified percentage away from southbound lane 5.

```
  From     Till    Duration    Trap #      Problem
8:48:00  8:54:00   0:06:00    4 - Up      Cross lanes OCC: S4, S5
```

AUXILIARY PARAMETERS: The one auxiliary parameter for this test is the percentage change allowable. The typical entry is:

```
Auxiliary parameters                    Default
LP_CROSS_OCC_RANGE_PERCENT                 15
```

LP_CROSS_ON_TEST This allows the user to specify the maximum percentage change in on time between lanes that is allowable.

OUTPUT IN ERROR FILE: The error file entry contains the two offending lanes in a two character code. The first character is either **S**, for southbound, or **N**, for northbound, and the second character is the lane number. In this example the on time in the southbound lane 4 was more than the specified percentage away from southbound lane 5.

```
  From     Till    Duration    Trap #      Problem
8:48:00   8:54:00   0:06:00    4 - Up      Cross lanes ON: S4, S5
```

AUXILIARY PARAMETERS: The one auxiliary parameter for this test is the percentage change allowable. The typical entry is:

```
Auxiliary parameters                    Default
LP_CROSS_ON_RANGE_PERCENT                  15
```

LP_CROSS_PPS_TEST This allows the user to specify the maximum percentage change in counts between lanes that is allowable.

OUTPUT IN ERROR FILE: The error file entry contains the two offending lanes in a two character code. The first character is either **S**, for southbound, or **N**, for northbound, and the second character is the lane number. In this example the counts in the southbound lane 4 was more than the specified percentage away from southbound lane 5.

```
  From     Till    Duration    Trap #      Problem
8:48:00   8:54:00   0:06:00    4 - Up      Cross lanes PPS: S4, S5
```

AUXILIARY PARAMETERS: The one auxiliary parameter for this test is the percentage change allowable. The typical entry is:

```
Auxiliary parameters                    Default
LP_CROSS_PPS_RANGE_PERCENT                 10
```

LP_CROSS_SPEED_TEST This allows the user to specify the maximum percentage change in speed between lanes that is allowable.

OUTPUT IN ERROR FILE: The error file entry contains the two offending lanes in a two character code. The first character is either **S**, for southbound, or **N**, for northbound, and the second character is the lane number. In this example the speed in the southbound lane 4 was more than the specified percentage away from southbound lane 5.

```
  From     Till    Duration    Trap #      Problem
8:48:00   8:54:00   0:06:00    4 - Up      Cross lanes SPEED: S4, S5
```

AUXILIARY PARAMETERS: The one auxiliary parameter for this test is the percentage change allowable. The typical entry is:

```
Auxiliary parameters                    Default
LP_CROSS_SPEED_RANGE_PERCENT               20
```

LP_OCC_HIGH_TEST This allows the user to specify the maximum value that the occupancy can take, and for how long.

OUTPUT IN ERROR FILE: This simply states that the value was exceeded.

```
  From     Till    Duration    Trap #      Problem
8:48:00   8:54:00   0:06:00    4 - Up      OCC passed high threshold.
```

AUXILIARY PARAMETERS: One auxiliary parameter for this test is the maximum value and the other is the number of times that it should pass this threshold before it is flagged.

```
Auxiliary parameters                    Default
LP_OCC_HIGH_THRESHOLD_PERCENT             60
LP_OCC_HIGH_THRESHOLD_NUM                  4
```

LP_OCC_LOW_TEST This allows the user to specify the minimum value that the occupancy can take, and for how long.

OUTPUT IN ERROR FILE: This simply states that the value was too low.

```
  From     Till    Duration    Trap #      Problem
8:48:00   8:54:00   0:06:00    4 - Up      OCC passed low threshold.
```

AUXILIARY PARAMETERS: One auxiliary parameter for this test is minimum value and the other is the number of times that it should pass this threshold before it is flagged.

```
Auxiliary parameters                    Default
LP_OCC_LOW_THRESHOLD_PERCENT              5
LP_OCC_LOW_THRESHOLD_NUM                  8
```

LP_OCC_NUM_ZEROS_TEST This allows the user to specify the maximum number of zeros that can occur in a row for the OCC value before it is flagged.

OUTPUT IN ERROR FILE: This simply states that the value was exceeded.

```
  From     Till    Duration    Trap #      Problem
8:48:00   8:54:00   0:06:00    4 - Down    OCC was zero too often.
```

AUXILIARY PARAMETERS: The one auxiliary parameter for this test is the maximum number of zeros that can occur in a row.

```
Auxiliary parameters                    Default
LP_OCC_NUM_ZEROS                          5
```

LP_OCC_RANGE_TEST This allows the user to specify the maximum percentage difference in the values of the upstream and downstream detectors for the OCC value. There is no threshold time associated with this test - the instant that this condition is detected it is flagged. Note that this is in terms of percentages.

OUTPUT IN ERROR FILE: This simply states that the value was exceeded.

```
  From     Till    Duration    Trap #      Problem
8:48:00   8:54:00   0:06:00    4 - Down    OCC passed range threshold.
```

AUXILIARY PARAMETERS: The one auxiliary parameter for this test is the maximum percentage difference in the values of the upstream and downstream detectors.

Auxiliary parameters	Default
LP_OCC_RANGE_PERCENT	10

LP_ON_TIME_CRITICAL_TEST This allows the user to specify the maximum value that the on time can ever take. If the on time exceeds this value just once then it is flagged. This is used to look for serious errors in a detector.

OUTPUT IN ERROR FILE: This simply states that the value was exceeded.

From	Till	Duration	Trap #	Problem
8:48:00	8:54:00	0:06:00	4 - Down	On time passed CRITICAL.

AUXILIARY PARAMETERS: The one auxiliary parameter for this test is the maximum critical value.

Auxiliary parameters	Default
LP_ON_TIME_CRITICAL	2000

LP_ON_TIME_TEST This allows the user to specify the maximum value that the on time can take, and for how long. This maximum value is in milliseconds.

OUTPUT IN ERROR FILE: This simply states that the value was exceeded.

From	Till	Duration	Trap #	Problem
8:48:00	8:54:00	0:06:00	4 - Up	On time passed threshold.

AUXILIARY PARAMETERS: One auxiliary parameter for this test is the maximum value and the other is the number of times that it should pass this threshold before it is flagged.

Auxiliary parameters	Default
LP_ON_TIME_HIGH_THRESHOLD_MSEC	300
LP_ON_TIME_HIGH_THRESHOLD_NUM	4

LP_PPS_HIGH_TEST - Warning: between match_output_period and per_sec This allows the user to specify the maximum value that the counts can take, and for how long. Note that if you change the value of **OUTPUT_FLOW_AVG_FACTOR** then you will affect the outcome of this test. Just remember that if you are generating count values that are in counts per second then it is never going to get above 2 or 3. But that if you generating count values in counts per output period then you can get very large numbers. You should set your tests appropriately.

OUTPUT IN ERROR FILE: This simply states that the value was exceeded.

From	Till	Duration	Trap #	Problem
8:48:00	8:54:00	0:06:00	4 - Down	PPS passed high threshold.

AUXILIARY PARAMETERS: One auxiliary parameter for this test is the maximum value and the other is the number of times that it should pass this threshold before it is flagged.

Auxiliary parameters	Default
LP_PPS_HIGH_THRESHOLD_COUNTS	10
LP_PPS_HIGH_THRESHOLD_NUM	4

LP_PPS_LOW_TEST This allows the user to specify the minimum value that the counts can take, and for how long. Note that if you change the value of the runfile parameter **OUTPUT_FLOW_AVG_FACTOR** then you will affect the outcome of this test. Just remember that if you are generating count values that are in counts per second then it is never going to get above 2 or 3. But that if you generating count values in counts per output period then you can get very large numbers. You should set your tests appropriately.

OUTPUT IN ERROR FILE: This simply states that the value was exceeded.

From	Till	Duration	Trap #	Problem
8:48:00	8:54:00	0:06:00	4 - Down	PPS passed low threshold.

AUXILIARY PARAMETERS: One auxiliary parameter for this test is the minimum value and the other is the number of times that it should pass this threshold before it is flagged.

Auxiliary parameters	Default
LP_PPS_LOW_THRESHOLD_COUNTS	1
LP_PPS_LOW_THRESHOLD_NUM	4

LP_PPS_NUM_ZEROS_TEST This allows the user to specify the maximum number of zeros that can occur in a row for the PPS value before it is flagged.

OUTPUT IN ERROR FILE: This simply states that the value was exceeded.

From	Till	Duration	Trap #	Problem
8:48:00	8:54:00	0:06:00	4 - Down	PPS was zero too often.

AUXILIARY PARAMETERS: The one auxiliary parameter for this test is the maximum number of zeros that can occur in a row.

Auxiliary parameters	Default
LP_PPS_NUM_ZEROS	5

LP_PPS_RANGE_TEST This allows the user to specify the maximum percentage difference in the values of the upstream and downstream detectors for the PPS value. There is no threshold time associated with this test - the instant that this condition is detected it is flagged. Note that this is in terms of percentages.

OUTPUT IN ERROR FILE: This simply states that the value was exceeded.

```
  From     Till    Duration    Trap #      Problem
8:48:00   8:54:00   0:06:00    4 - Down    PPS passed range threshold.
```

AUXILIARY PARAMETERS: The one auxiliary parameter for this test is the maximum percentage difference in the values of the upstream and downstream detectors.

```
Auxiliary parameters                    Default
LP_PPS_RANGE_PERCENT                      10
```

LP_SPEED_HIGH_TEST This allows the user to specify the maximum value that the speed can take, and for how long. This maximum value is in miles per hour.

OUTPUT IN ERROR FILE: This simply states that the value was exceeded.

```
  From     Till    Duration    Trap #      Problem
8:48:00   8:54:00   0:06:00    4           SPEED passed high threshold.
```

AUXILIARY PARAMETERS: One auxiliary parameter for this test is the maximum value and the other is the number of times that it should pass this threshold before it is flagged.

```
Auxiliary parameters                    Default
LP_SPEED_HIGH_THRESHOLD_MPH               65
LP_SPEED_HIGH_THRESHOLD_NUM                5
```

LP_SPEED_LOW_TEST This allows the user to specify the minimum value that the speed can take, and for how long. This minimum value is in miles per hour.

OUTPUT IN ERROR FILE: This simply states that the value was exceeded.

```
  From     Till    Duration    Trap #      Problem
8:48:00   8:54:00   0:06:00    4           SPEED passed low threshold.
```

AUXILIARY PARAMETERS: One auxiliary parameter for this test is the minimum value and the other is the number of times that it should pass this threshold before it is flagged.

```
Auxiliary parameters                    Default
LP_SPEED_LOW_THRESHOLD_MPH                20
LP_SPEED_LOW_THRESHOLD_NUM                 5
```

10.3 The Default Values For The Loop Tests

All of the defaults for the tests are **NO**, meaning that they are turned off. These are listed here in Table 10.1 only for completeness (not because I wanted to make the manual any longer).

All of the default values for the auxiliary parameters are listed out in Table 10.2. Also, listed out in Table 10.3 are the main parameters for the tests and the entries that they place in the error file. The characters XN means the direction and the lane number.

Parameter	Default
LP_CROSS_OCC_TEST	NO
LP_CROSS_ON_TEST	NO
LP_CROSS_PPS_TEST	NO
LP_CROSS_SPEED_TEST	NO
LP_OCC_HIGH_TEST	NO
LP_OCC_LOW_TEST	NO
LP_OCC_NUM_ZEROS_TEST	NO
LP_OCC_RANGE_TEST	NO
LP_ON_TIME_CRITICAL_TEST	NO
LP_ON_TIME_TEST	NO
LP_PPS_HIGH_TEST	NO
LP_PPS_LOW_TEST	NO
LP_PPS_NUM_ZEROS_TEST	NO
LP_PPS_RANGE_TEST	NO
LP_SPEED_HIGH_TEST	NO
LP_SPEED_LOW_TEST	NO

Table 10.1: Test parameter defaults.

Auxiliary Parameter	Default
LP_CROSS_OCC_RANGE_PERCENT	15
LP_CROSS_ON_RANGE_PERCENT	15
LP_CROSS_PPS_RANGE_PERCENT	10
LP_CROSS_SPEED_RANGE_PERCENT	20
LP_OCC_HIGH_THRESHOLD_NUM	4
LP_OCC_HIGH_THRESHOLD_PERCENT	60
LP_OCC_LOW_THRESHOLD_NUM	8
LP_OCC_LOW_THRESHOLD_PERCENT	5
LP_OCC_NUM_ZEROS	5
LP_OCC_RANGE_PERCENT	10
LP_ON_TIME_CRITICAL	2000
LP_ON_TIME_HIGH_THRESHOLD_MSEC	300
LP_ON_TIME_HIGH_THRESHOLD_NUM	4
LP_PPS_HIGH_THRESHOLD_COUNTS	10
LP_PPS_HIGH_THRESHOLD_NUM	4
LP_PPS_LOW_THRESHOLD_COUNTS	1
LP_PPS_LOW_THRESHOLD_NUM	4
LP_PPS_NUM_ZEROS	5
LP_PPS_RANGE_PERCENT	10
LP_SPEED_HIGH_THRESHOLD_MPH	65
LP_SPEED_HIGH_THRESHOLD_NUM	5
LP_SPEED_LOW_THRESHOLD_MPH	20
LP_SPEED_LOW_THRESHOLD_NUM	5

Table 10.2: Auxiliary parameter defaults.

Main Parameter	Error Entry
LP_CROSS_OCC_TEST	Cross lanes OCC: XN, XN
LP_CROSS_ON_TEST	Cross lanes ON: XN, XN
LP_CROSS_PPS_TEST	Cross lanes PPS: XN, XN
LP_CROSS_SPEED_TEST	Cross lanes SPEED: XN, XN
LP_OCC_HIGH_TEST	OCC passed high threshold.
LP_OCC_LOW_TEST	OCC passed low threshold.
LP_OCC_NUM_ZEROS_TEST	OCC was zero too often.
LP_OCC_RANGE_TEST	OCC passed range threshold.
LP_ON_TIME_CRITICAL_TEST	On time passed CRITICAL.
LP_ON_TIME_TEST	On time passed threshold.
LP_PPS_HIGH_TEST	PPS passed high threshold.
LP_PPS_LOW_TEST	PPS passed low threshold.
LP_PPS_NUM_ZEROS_TEST	PPS was zero too often.
LP_PPS_RANGE_TEST	PPS passed range threshold.
LP_SPEED_HIGH_TEST	SPEED passed high threshold.
LP_SPEED_LOW_TEST	SPEED passed low threshold.

Table 10.3: Main parameters and error entries.

Chapter 11

Program Input: Cross Data Analysis

One goal of the **fsp** program is to calculate the delay for each specific incident. In order to do this, quite a few things need to be done. This chapter will explain in detail all of the steps that were taken to calculate the incident delay and all of the formulas that were used.

I would like to start out with an overview of how the data is processed. In the following sections I'll explain each step in detail.

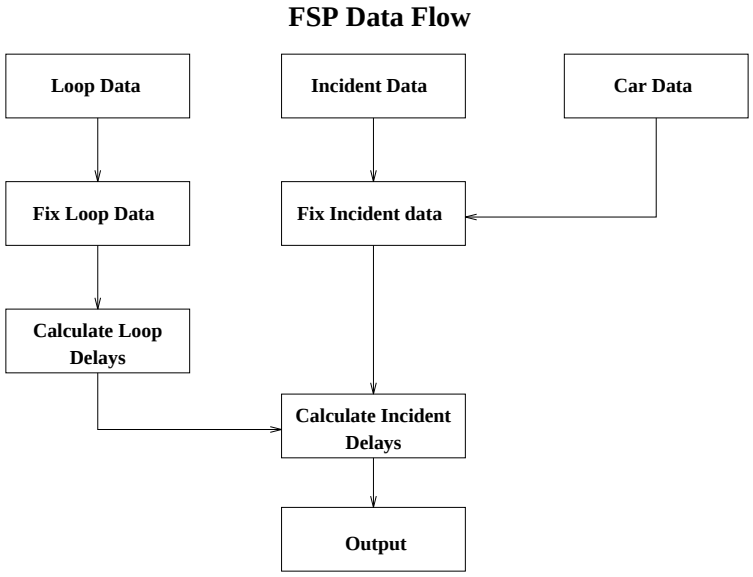

Figure 11.1: Big Picture For FSP Program.

Figure 11.1 is a rough view of what the data flow looks like in the **fsp** program. There are three main branches of the program: the loop data, the incident data, and the car data. Since the overall goal is to calculate the delay per incident the loop and car branches merge into the incident data branch at various points. The car data is used mainly to correct the location and duration of each incident. The loop data is then used to calculate the delay

per incident. The loop data processing is done in two main steps. First we fix the loop data and then we generate the loop delay files. The things that we try to fix in the loop data are the various holes that show up and the consistency errors. Both of these fixes are discussed in Chapter 5. The loop delay files are simply the delay calculated for each loop for every time segment. The car data is first read in to generate some diagnostic graphs and then it is used to fix various fields in the incident database. The incident data processing involves first filtering the incidents, then fixing their locations, and finally calculating the delay for each incident. At each step in the data flow there are many different types of plots and tables that are made to explain what is happening. These are not shown in Figure 11.1 because it would obscure the simple message that I am trying to convey. Below is a list of steps that the program will take when calculating the delay per incident:

- Loop data processing
    1. Convert from raw data to ascii.
    2. Fix the loop data.
    3. Compute the loop averages.
    4. Compute the loop delays.
- Car data processing
    1. Convert from raw data to ascii.
    2. Record all key presses.
    3. Record all INRAD points.
- Incident data processing
    1. Filter out the proper incidents.
    2. Fix the incident locations from the car data.
    3. Fix the incident durations from the car data.
- Calculate the delay per incident
    1. Read in the incident bounding boxes.
    2. Read in the loop files corresponding to those boxes.

In the following sections I will discuss what processing goes on in each of the boxes in Figure 11.1 and in each of the steps in the list above.

11.1 Generating The Loop Speeds

It turns out that there are a couple of different ways to calculate the delay on a section of the freeway. The standard equation for calculating the delay at a particular loop segment is:

$$D = L\frac{\Delta T}{60}F\left(\frac{1}{V} - \frac{1}{V_T}\right) \tag{11.1}$$

Where D is the delay for a particular segment of the freeway, L is the length, F is the flow, V is the speed, and V_T is the speed of congestion. Even though this is pretty straight forward it is not clear exactly how to apply this formula. Since we have the flows and the speeds for each lane at every loop segment should we calculate the delay for each lane and then add them up to get a total delay for each segment? Or should we average the flows and the speeds over the lanes and then calculate the delay? Well, if we look at these two different methods then we will see that they are not the same. The delay that we get by first calculating the delay for each lane and then adding up all of the lanes (assuming that there are N lanes) is the following:

$$D_i = L\frac{\Delta T}{60}F_i\left(\frac{1}{V_i} - \frac{1}{V_T}\right) \tag{11.2}$$

$$D = \sum_{i=1}^{N} D_i \tag{11.3}$$

$$D = \sum_{i=1}^{N} L\frac{\Delta T}{60}F_i\left(\frac{1}{V_i} - \frac{1}{V_T}\right) \tag{11.4}$$

$$D = L\frac{\Delta T}{60}\sum_{i=1}^{N}\left(\frac{F_i}{V_i} - \frac{F_i}{V_T}\right) \tag{11.5}$$

Note that anything indexed by i is an individual lane. These equations have been carried out this far so that we can compare them to the second way of calculating the delay which is to first compute the average flow and speed over the freeway and then to just do one delay calculation:

$$F = \sum_{i=1}^{N} F_i \tag{11.6}$$

$$V = \frac{\sum_{i=1}^{N} F_i V_i}{\sum_{i=1}^{N} F_i} \tag{11.7}$$

$$D = L\frac{\Delta T}{60}F\left(\frac{1}{V} - \frac{1}{V_T}\right) \tag{11.8}$$

$$D = L\frac{\Delta T}{60}\left(\sum_{i=1}^{N} F_i\right)\left(\left(\frac{\sum_{i=1}^{N} F_i}{\sum_{i=1}^{N} F_i V_i}\right) - \frac{1}{V_T}\right) \tag{11.9}$$

As you can see, equation 11.5 is not the same as equation 11.9. So the question arises, "which one represents the proper delay calculation?" The solution that the **fsp** program implements is the first: conceptually it figures out the delay for each lane and then figures out the total delay by summing up over all of the lanes. It does it conceptually because it never actually calculates the delay for each lane. Instead, the **fsp** program does a calculation like equations 11.6 thru 11.9 but it uses a different type of average for the speed. It uses the weighted harmonic average speed instead of the arithmetic speed. If we do this we get a total speed for the freeway as:

$$V_h = \frac{\sum_{i=1}^{N} F_i}{\sum_{i=1}^{N} \frac{F_i}{V_i}} \tag{11.10}$$

Plugging this average into equation 11.1 we get:

$$D = L\frac{\Delta T}{60}F\left(\frac{1}{V_h} - \frac{1}{V_T}\right) \tag{11.11}$$

$$D = L\frac{\Delta T}{60}\left(\sum_{i=1}^{N} F_i\right)\left(\left(\frac{\sum_{i=1}^{N}\frac{F_i}{V_i}}{\sum_{i=1}^{N} F_i}\right) - \frac{1}{V_T}\right) \tag{11.12}$$

$$D = L\frac{\Delta T}{60}\sum_{i=1}^{N}\left(\frac{F_i}{V_i} - \frac{F_i}{V_T}\right) \tag{11.13}$$

This is the same as equation 11.5. There are two reasons for wanting to calculate the delay for the segment by using equations of the form 11.6 thru 11.9:

- In order to fix the holes at all we need to have one value for the speed and one value for the flow at the detectors adjacent to each hole. Therefore, we need the averages over all the lanes for all of the loop detectors. Finally, since we can't do the individual lane calculations on the fixed data and since we already have the averages we might as well do the speed calculation as in equation 11.10.
- Equations 11.6 thru 11.9 are more computationally efficient than equations 11.2 thru 11.5.

The whole point of the previous discussion is that the average loop speed files contain the weighted harmonic average instead of the arithmetic average. In normal traffic flow conditions these two speeds shouldn't vary by more than a few percent. But when the spread in the speed between the lanes is high, the arithmetic and weighted harmonic averages can vary by up to 20%. Usually, the only time when the speeds across the lanes will vary significantly is when there is an incident on a section of the freeway where there is a high occupancy vehicle (HOV) lane. Since cars won't change into the HOV lane to avoid the congestion for fear of a fine, the traffic in the non-HOV lanes tends to build up rather quickly and hence the speeds across the lanes starts to vary.

11.2 Fixing The Loop Data

The first thing that needs to be done after the speeds have been calculated and before any analysis can take place is to fix the loop data. As was discussed in Section 5.2, the loop data has a number of things wrong with it. The **fsp** program will try to make two main fixes: a hole fix and a consistency fix. I have referred to these as the various stages of the loop data processing. Figure 11.2 shows the loop data flow.

Each box produces a different type of loop file. There is a short synopsis of each file type below. The sections that follow explain each step in detail.

Raw loop data The raw loop data is an encoded file that holds the data stream from the 170 controller. The files have names like: `loop5.dat`.

Extract to ascii text These files are separated according to lane number and data type. This means that each lane and each type of data (flows, occupancies, or speeds) is in it's own file. A typical file name from this set of loop files would be: `floop3.nc2`. This corresponds to the ascii text file from loop detector #3 and the northbound flows from lane 2. These files are called the "floop" files and are referred to as the first stage in the loop processing. For more information on the naming scheme of the loop output files see Chapter 15.

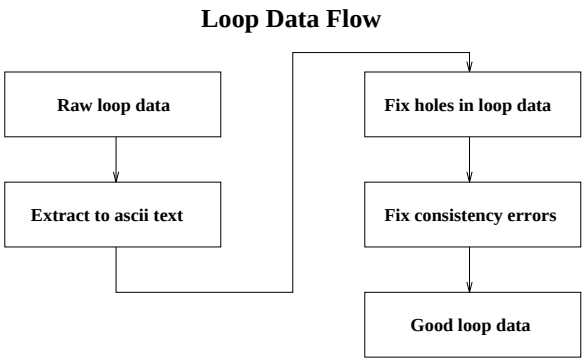

Figure 11.2: Fixing The Loop Data.

Fix holes in loop data These files are exactly like the "floop" files except that the holes have been fixed. These files have filenames that start with "gloop" and hence are referred to as the "gloop" files. These are generated in the second stage of the loop processing.

Fix consistency errors These files are the same as the "gloop" files except that the consistency errors have been corrected. These files have filenames that start with "hloop" and hence are referred to as the "hloop" files. This is the third and final stage of the loop processing.

So when the program gets done processing the loop data there could be three different sets of loop files on your system: floop, gloop, and hloop. The **fsp** program gives you the option of deleting the sets of loop files that you don't need. This is done via the runfile parameters FLOOP_CLEANUP, GLOOP_CLEANUP, and HLOOP_CLEANUP.

11.2.1 The floop Files

We start off with the raw loop data and the first thing that we do is convert the data from raw form to ascii form. This is called the first stage and it generates a bunch of files called "floop" files. The raw data has one file for each loop detector that contains all of the counts, speeds, and occupancies for every lane. For example, let's say that we are looking at loop detector #1. Then the raw loop data file will be `loop1.dat`. What the first stage of processing does is it extracts into separate files the data for each lane and for each on and off ramp at that detector, and it does this for the counts, speeds, and occupancies at each lane. Since detector #1 has 5 southbound and 5 northbound lanes the first stage would generate a total of 36 files for just that detector. These are listed out below:

```
oriel 1: ls floop1.*
floop1.nc1      floop1.ns1      floop1.so1
floop1.nc2      floop1.ns2      floop1.so2
floop1.nc3      floop1.ns3      floop1.so3
floop1.nc4      floop1.ns4      floop1.so4
floop1.nc5      floop1.ns5      floop1.so5
floop1.ncd      floop1.nsd      floop1.sod
```

```
floop1.no1      floop1.sc1      floop1.ss1
floop1.no2      floop1.sc2      floop1.ss2
floop1.no3      floop1.sc3      floop1.ss3
floop1.no4      floop1.sc4      floop1.ss4
floop1.no5      floop1.sc5      floop1.ss5
floop1.nod      floop1.scd      floop1.ssd
```

As you can see, the reason that these are called the "floop" files is because they all start with the prefix "floop." You can also see that this is quite a few files for just one loop detector. The naming scheme for these files is: floopWW.XYZ. Where:

floop: this is just the standard file prefix.

WW: this is the loop detector number (or cabinet number).

X: this is either "n" or "s" for the northbound or southbound direction.

Y: explains the type of data and can be one of the following:

c: means counts or pps.

s: means speed.

o: means occupancy.

Z: this is the lane number or the on or off ramp number.

For example, the file `floop1.ns4` is the raw speed data for lane 4 of the northbound direction of detector # 1. If the last character is not a number but a "d" then that means the file is the value of the average of all of the lanes. There are a couple of important things to note about the naming of the loop files but that would take us too far astray. The complete discussion can be found in Section 15.3.

11.2.2 The gloop Files

Once the "floop" files have been extracted from the raw data the program can attempt to fix the holes that appear in the data. This is called the second stage and it will generate a bunch of files called the "gloop" files. The "gloop" files are just the "floop" files with the holes filled in. If it turns out that there aren't any holes in a particular "floop" file then that file is simply copied over to the appropriate "gloop" file. See the discussion in Section 5.2.1 for a complete explanation of how the loop data is recreated. The naming scheme for both sets of files is the same. For example, `floop1.sod` and `gloop1.sod` both refer to the average southbound occupancy at detector # 1. It's just that the first file is the raw data and could have holes in it, whereas the second file is the corrected data and hence won't have any holes at all. There are a couple of important things to note about the hole fix algorithm that generates the "gloop" files:

- The algorithm only operates on the average files, not on the individual lane files. Subsequently, there is no way to fix the individual lane files.

- The algorithm combines all of the on ramp files into a single file depending on type. So all of the on ramp occupancy files are combined to give a single on ramp occupancy file. The same holds for the flow files on the on ramps.
- The algorithm combines all of the off ramp files into a single file depending on type exactly like the on ramp files.
- See the discussion under the runfile parameter `LOOP_HOLES_FIX` for a discussion of what other parameters need to be set in order for this fix to run successfully.

If you are generate the "gloop" files then you probably don't need any of the "floop" files on the system anymore. You can tell the program to delete them when it's done with them by setting the runfile parameter `FLOOP_CLEANUP` to be `DELETE_EVERYTHING` This is discussed in Chapter 7.

11.2.3 The hloop Files

Finally, the program will attempt to fix the consistency errors in the loop data. The algorithm can only fix the consistency errors in the "gloop" files. This means that in order to fix the consistency errors that the user has to first run the hole fixing algorithm. The consistency fix algorithm itself is described in Section 5.2.2. The files that this fix generates are the same as the "gloop" files in terms of filenames except that there is an "h" where there used to be a "g" and hence these are referred to as the "hloop" files.

Once you generate the "hloop" files you can have the program delete the "floop" and "gloop" files by setting the runfile parameters `FLOOP_CLEANUP` and `GLOOP_CLEANUP` to `DELETE_EVERYTHING`.

11.3 The Loop Delay Files

After the fixes have been done on the loop data the next major step that the program takes is to calculate the loop delay files. The loop delay algorithm can take in any one of the loop sets, either the "floop," "gloop," or the "hloop" files. I would recommend that you only run this on the "gloop" or "hloop" files because there are so many holes in the "floop" data.

When we say that we are going to calculate the loop delay we mean that we are going to figure out the following value:

$$D_k^i = L\frac{\Delta T}{60}F_k^i\left(\frac{1}{V_k^i} - \frac{1}{V_T}\right) \tag{11.14}$$

Where D_k^i is the delay on segment k during time slice i, L is the segment length in miles, ΔT is the time slice in minutes, F_k^i is the flow on segment k during time slice i, V_k^i is the speed on segment k during time slice i, and V_T is the threshold or congestion speed[1]. For each loop detector we calculate this value for every time slice. The resulting files are called the loop delay files.

[1]Note that the indices here mean something different than in equation 11.2.

11.3.1 The Runfile Parameters Needed

The loop delay files are the basis of the delay per incident calculation. The routine in the program that calculates the delay per incident reads in the loop delay files to figure out the delay surrounding an incident. You should be aware that the parameters that you set when calculating the loop delay files also effect the delay per incident calculation. There are a couple of runfile parameters that you can set that govern the generation of the loop delay files. These are all explained in Chapter 7 but are listed here for convenience. These are:

TRAFFIC_DELAY Whether or not to calculate the loop delay.

DELAY_CALCULATION Specify whether to calculate the delay with respect to a constant congestion speed or the average.

TRAFFIC_LOW_SPEED Specify what the congestion speed should be.

DELAY_TYPE Specify whether to allow negative delays or not.

The parameter `TRAFFIC_DELAY` simply tells the **fsp** whether to run the routine that calculates the loop delay files. In the overall flow of the **fsp** program the loop data is processed first followed by the incident data. When the incident data is processed, it needs to read in the loop delay files. Well, these files are not stored in the computer memory, they are stored in specific directories. So it doesn't matter if the loop delay files were generated by this run of the **fsp** program or a run that was done last week - the files will still be in the directories. If you had to calculate the delay files every time that you wanted to do something different with the incidents then the program would take way too much time. By turning off the calculation of the loop delay files and just using the ones that are on the hard disk you save quite a bit of time. Of course there are problems with this:

- If somebody deletes the loop delay files and the routine that processes the incidents can't find them then the program will halt.
- If you want to change ANY of the parameters that governed the generation of the loop delay files then you have to recalculate them. This includes, but is not limited to: the congestion speed, the relevant time period, the output period, the loop filter factor, etc.

The parameter `DELAY_CALCULATION` is probably the most important runfile parameter involved in calculating the delays. This parameter tells the program to calculate the delays with respect to a constant congestion speed or with respect to an average congestion speed. This is the same as setting V_T in equation 11.14 to a constant or to an average speed. For a constant congestion speed this can be seen in Figure 11.3. This is a typical speed vs. time plot for a single loop detector. Note that the delay here can be positive or negative - this can be changed so that the delay can only be positive. On our study section there is recurrent congestion at a particular time of day at a particular location and we didn't want this congestion to be counted as delay. We decided that if we find the average speed over all of the days that this would give us a pretty good measure of where the recurrent congestion was taking place. If the congestion speed is taken to be the average then the picture will look more like Figure 11.4. In order to use the average speeds you need to first calculate them. In Chapter 12 there is a

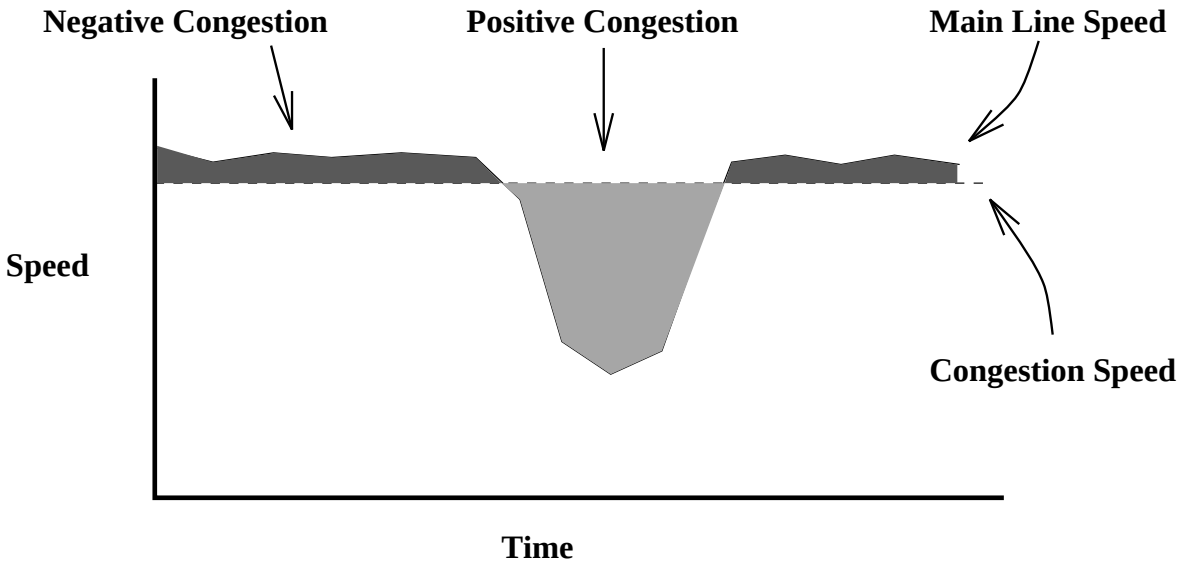

Figure 11.3: Delay Calculation wrt A Constant.

complete example of how to calculate the average speeds and how to then calculate the loop delay files and the delay per incident.

If the runfile parameter `DELAY_CALCULATION` is set to indicate that the congestion speed should be a constant, then the program will get the congestion speed from the parameter `TRAFFIC_LOW_SPEED`.

The parameter `DELAY_TYPE` tells the program whether or not to allow negative delays. If the value of V_T is larger than the value of V_k^i in equation 11.14 then the delay can be negative. Since it is not known whether it is meaningful for the delay to be negative or not we leave this up to the user. If delays are not allowed to be negative then all negative delays are set to zero.

11.3.2 Extra Loop Files

Even though I don't want to lead the discussion too far astray, I need to mention that there are some other files that are generated, or can be generated, when the program is calculating the loop delay files. On the first reading the reader can skip to Section 11.4 without any loss of continuity.

The Loop Delay Tables: For each time slice a table is made that holds the various parameters for the delay calculation. These tables are placed into a LaTeXfile that can be processed to produce a postscript document that can be printed to any printer. The details on how to do this are given in Chapter 13 with the file naming conventions given in Chapter 15.

The Cumulative Loop Delay Files: For each time period a file is made that is the cumulative loop delay for the whole freeway. This is discussed in Chapter 15.

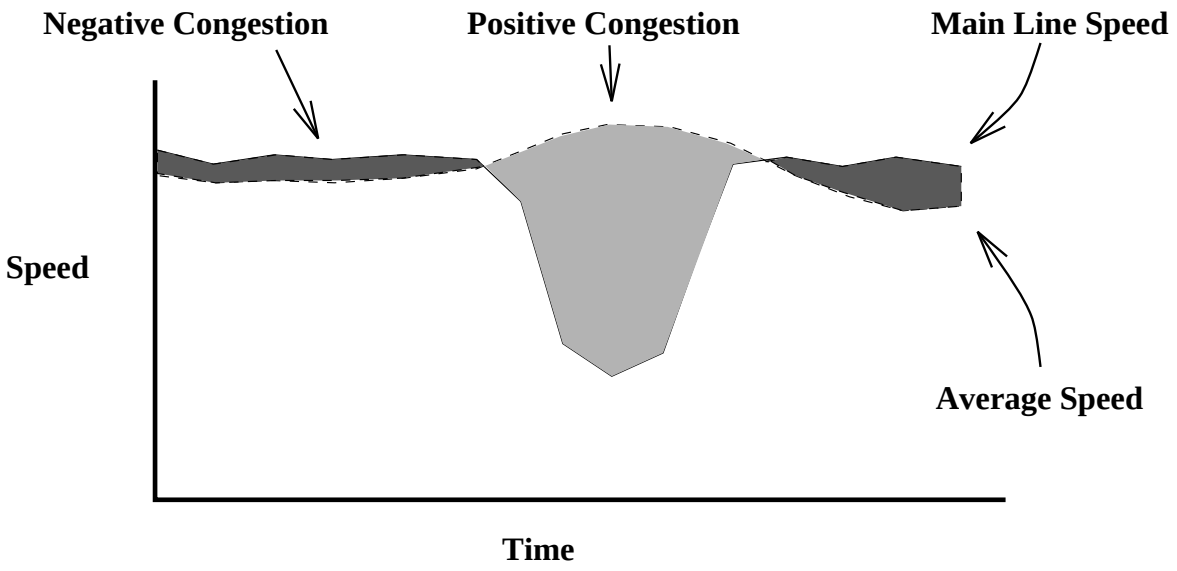

Figure 11.4: Delay Calculation wrt The Average.

The Loop Emission Files: The amount of emissions produced on the freeway is calculated for each loop segment and each time slice. The emissions calculated are for hydrocarbons, nitrogen compounds, and carbon monoxide. Once again, these are discussed in Chapter 15.

11.4 Fixing The Incident Data

In addition to processing the loop files to generate the good loop data, the program also needs to process the car data in order to fix various fields in the incident database. Since the processing of the car data and the fixing of the incident data are so closely tied together, I will discuss them in the same section.

When processing the incident data the first step that is taken is to filter the incident data. The format of the incident filter is discussed in Chapter 9. Since the filtering of the incidents is pretty straight forward we will assume in this section that the incidents have already been filtered. A more detailed look at the data flow for fixing the incidents is given in Figure 11.5.

There are two things that the program tries to fix about the incidents: the incident position and the incident duration. The box labeled "Fix incident placement" in Figure 11.5 has three different arrows pointing to it. The arrow in the middle simply represents the incidents being passed from the routine that filters out the correct incidents. The two other arrows come from the car data and a runtime file. These are meant to represent that the data for the incident location fix can come directly from the car data or from a runtime file. Since processing the car data takes so long what we prefer to do is to process the car data once and then save the results in a file. This file can then be read in at runtime instead of processing the car data. This saves a considerable amount of time. The box labeled "Fix incident duration" does almost the same

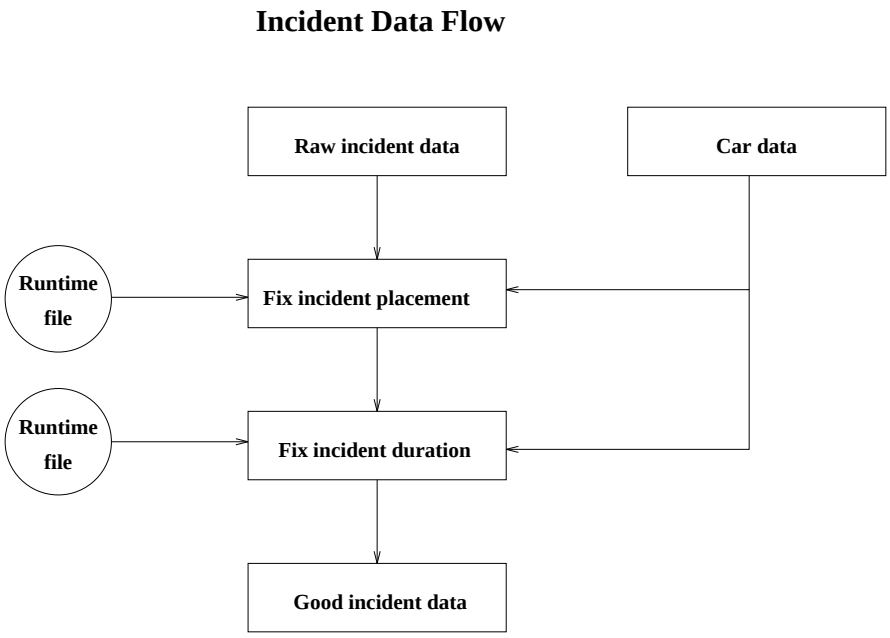

Figure 11.5: Data Flow For Fixing The Incidents.

thing as the "Fix incident placement" box except that it fixes the incident durations instead of their locations.

Either of these fixes can use the car data or the runtime file independently of the other. There are some runfile parameters that you have to set if you want to use the car data to fix the incident data. The algorithms used to fix the incident data are explained in detail in Sections 5.3.1 and 5.3.2 and examples of doing each of these fixes are given in Chapter 12. We will assume that the runtime files are going to be used from now on.

There are so many files that are generated during the processing of the car data that I won't even give a quick summary here. A complete list, with examples, is given in Chapter 14.

11.5 Finding The Delay For Each Incident

The final goal of all of this processing is to figure out the delay for each incident. The data flow of the incident delay calculation looks like Figure 11.6. The routine takes in the loop delay files and filtered incidents and then calculates the delay per incident.

11.5.1 Incident Delays By Distance

On way to calculate the incident delay is to simply sum up the delay over the adjacent loop detectors for the time period of the incident. This can be represented with the following equation:

$$D_{incident} = \sum_{i \in ADJ} \sum_{j \in [T_s, T_e]} D_i^j \tag{11.15}$$

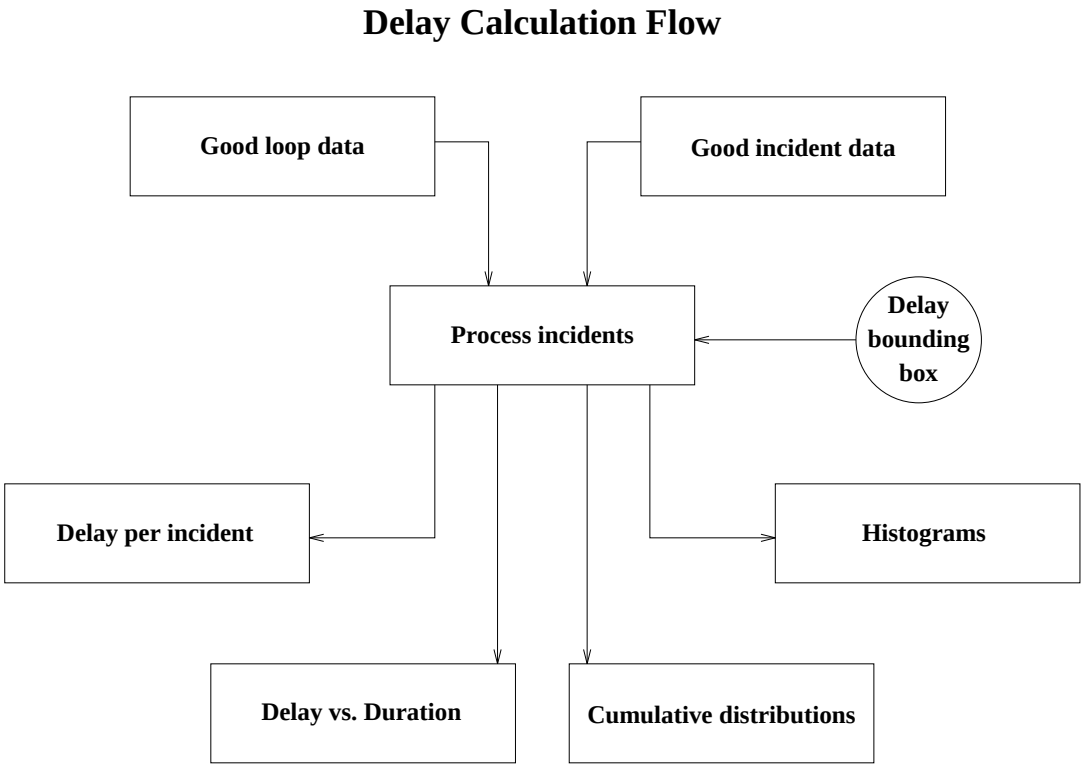

Figure 11.6: Processing The Incidents.

Where ADJ is the set of adjacent loop detectors defined below, T_s is the incident start time, T_e is the incident end time and D_i^j is the delay on segment i during time slice j. This is probably the most straight forward way of calculating the delay per incident. The set of adjacent loop detectors, or ADJ in formula 11.15, is defined by the following runfile parameters:

DELAY_UPSTREAM_NUM: This parameter tells the program the number of upstream loop detectors that you want to include in the delay calculation. A value of -1 indicates that you want to go all of the way back to the beginning of the study section. Note that the current loop detector is always included.

DELAY_DOWNSTREAM_NUM: This parameter tells the program the number of downstream loop detectors that you want to include in the delay calculation. A value of -1 indicates that you want to go all of the way down to the end of the study section. This is usually set to zero to indicate that you don't want to look at any detectors downstream.

For example, if `DELAY_UPSTREAM_NUM` was set to 3 and `DELAY_DOWNSTREAM_NUM` was set to 1 then the 3 upstream detectors and the first downstream detector from the incident site, for a total of 5 detectors, would be used in the delay calculation. Figure 11.7 is a picture of a typical speed vs. distance plot for a single time slice. In this picture, there was an incident that occurred at detector #6. We can see that the incident caused traffic to slow down upstream of the incident for approximately 4 detectors. We should probably include the upstream detectors 20, 9, 2 and 11, and the downstream detector 18 in our calculation of the delay for this incident.

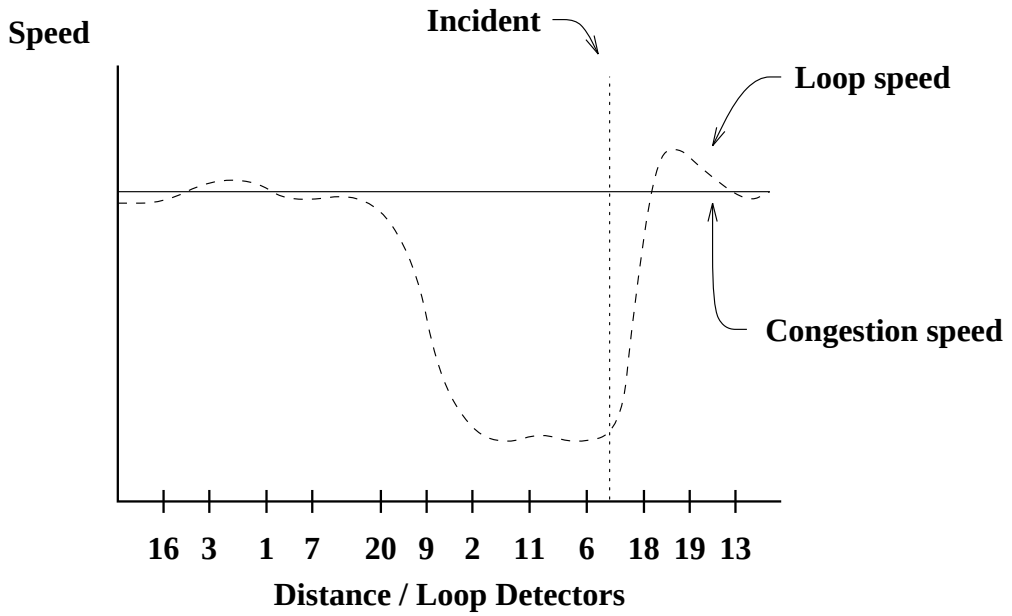

Figure 11.7: Incident At One Time Slice.

The main problem this method of incident delay calculation is that the number of detectors to include in the set ADJ is fixed for all of the incidents. When you tell the **fsp** program that you want to go upstream 4 detectors to calculate the delay then this means you want to do that for every incident. This is obviously going to be a problem. If there are two incidents of different duration during different periods of traffic flow then there is no reason to expect that the length of the queue buildup would be the same. One possible way to get around this is to use all of the upstream detectors when calculating the delay. But if there are multiple incidents during the same time period then you will be double counting the delay from one of the incidents. The way around these problems is to perform the incident delay calculation a different way.

11.5.2 Incident Delays By Bounding Box

There is a different way to calculate the delay per incident that is specific to each incident. This is done by figuring out where the effect of each incident ends and defining a bounding box around this region. This is done in a few steps:

1. Calculate the density for each loop detector for each time slice - exactly like the loop delay but for traffic density.
2. Make a contour plot of the traffic density for each shift with the incidents plotted on top.
3. Determine from the contour plot how far upstream the effect of the incident is felt. This can be done by figuring out where the density returns to normal.
4. Determine how long the incident has an effect on the density. This can be done by looking for the time that the density returns to normal.

5. These parameters form a box in time-space coordinates. Save these parameters, for each incident, to a file.

6. Read this file in at runtime and calculate the delays only over that bounding box.

An example of this can be seen in Figure 11.8. Figure 11.8 is a plot of the differential density

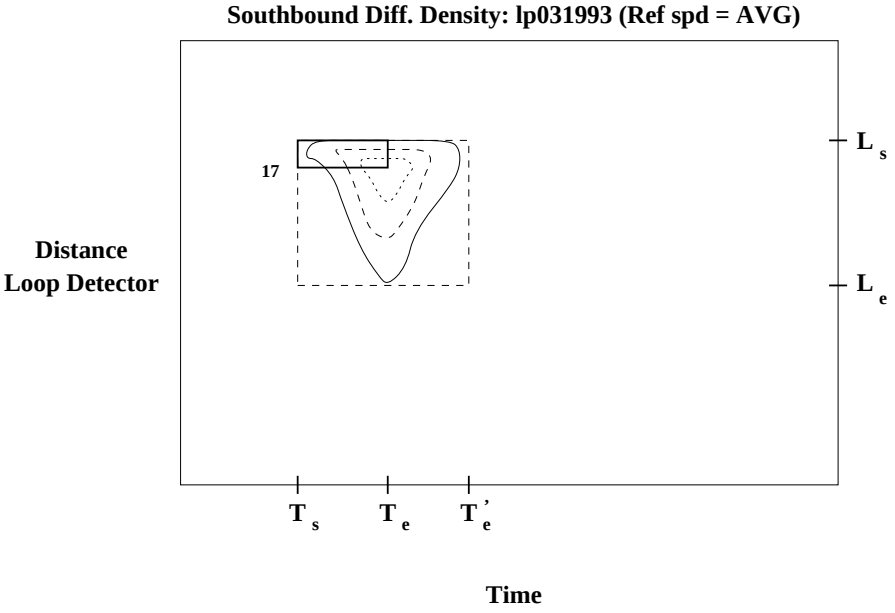

Figure 11.8: Density Contour With Incident.

over distance and time. The differential density is the density for a specific day divided by the average density for the whole study period. This specify plot is for the southbound section on 3/19/93. The only incident plotted on this graph is incident #17. The dark solid box is the time that we think the incident occurred. This duration could be the duration in the incident database or it could be the fixed duration - it depends on what parameters you specified in the runfile. The height of the box has no meaning - it is just there so that you can see the incident. The midpoint of the box is centered where we believe the incident took place. As you can see, there are some contours that start right when our incident starts and then build upstream. Once the incident is cleared the density starts to dissipate. The dotted box on the plot is the bounding box that should be defined for this incident. Note that it completely covers the bubble in density that was caused by this incident. With this new information the program can calculate the delay for this incident in step 6 above as:

$$D_{incident} = \sum_{i \in [L_e, L_s]} \sum_{j \in [T_s, T'_e]} D_i^j \tag{11.16}$$

The bounding box file that is read in at runtime is shown as the circle in Figure 11.6. The runfile parameter that you need to set to perform this type of incident delay calculation is `FIX_INC_DELAY_BOX`. This parameter has two possible values:

NO_FIX_INC_DELAY: This will cause the method described in Section 11.5.1 to be used to calculate the incident delay.

YES_FIX_INC_DELAY: This will tell the program to read in the bounding box at runtime and to use equation 11.16 to calculate the delay per incident.

Once again, there are a number of problems with using this method:

- The bounding boxes have to be square. There is no fundamental reason for this other than programming simplicity.
- If an incident doesn't have a bounding box defined in the runtime file then the delay for that incident is assumed to be zero.
- Even when inspecting the density plots by hand it is hard to separate multiple incidents. At present, there is no way to deal with multiple incidents where the congestion, or the queues, overlap.

Chapter 12

Examples With The Runfile

Since manipulating the runfile can be pretty daunting I have included some helpful hints and quite a few examples. The hints first:

- Always start with another runfile. There are usually too many parameters to type in for you to type in everything every time.
- Don't delete lines in the runfile, just comment them out. This will allow you to switch back and forth between different options really quickly and you won't have to remember all of those parameter names. The comment character is the "#" symbol.
- If the program can't read in a line in the runfile then try deleting the line entirely and re-typing it. Sometimes control characters can hide themselves in lines.
- You can not have more than 500 lines in the runfile. If you need more lines then you have to change the defined variable `MAX_NUM_RUNFILE_STRINGS` in the file `fsp.h` to be larger and then recompile the program and run it again.
- You have to have an equals sign in each one of the lines, even if you are just going to use the default value.
- If you are trying to generate error reports for the loop data and nothing is coming out then make sure that you have specified some tests to run. Remember that the default for all of the loop tests is **OFF**.
- If you think that some parameter is not being read in then you can put the program in preliminary debug mode. This is done by placing an extra "1" on the end of the command line. Something like:

```
clair 10: fsp my.runfile my.inc.filter 0 1
```

  This will simply print out all of the parameters that the program finds in the runfile.

12.1 General Parameters

There are a few general parameters to the runfile that are usually never changed. I will list these out here and then assume that they are always specified in the following examples. Almost all of the parameter defaults are either "NO" or "OFF," whichever is appropriate. So if a parameter is ever left out of a runfile then it is assumed to be set to the default. The following is only a part of a runfile. You could say that it's the runfile header because it doesn't do anything by itself - it only sets up boring general parameters:

```
# This is my boring runfile header.          (1)
GNU_PRINTER          = lw273                  (2)
ERROR_FILE_NAME_EXT  = err                    (3)
NAV_DATA_FILE_NAME   = nav.dat                (4)
KEY_DATA_FILE_NAME   = key.dat                (5)
CAR_DIRECTORY_ROOT   = car                    (6)
REPORT_DESTINATION   = FILE                   (7)
DEBUG_LEVEL          = SILENT_DEBUG           (8)
#DEBUG_LEVEL         = MINIMUM_DEBUG          (9)
#DEBUG_LEVEL         = DETAIL_DEBUG           (10)
#DEBUG_LEVEL         = VERBOSE_DEBUG          (11)
```

The numbers after each line are not part of the runfile - they are just there for reference purposes. A line by line explanation of the above runfile follows:

1) This is a comment. It is ignored by the program.

2) This specifies which printer all of the plots should go to. This string is placed directly in the gnuplot executable files.

3) This will specify the extension for the car error files.

4) This will specify the name of the file that holds the digital compass data.

5) This will specify the name of the file that holds the key presses from the probe vehicles.

6) This will specify the prefix of the directory names that hold the car data.

7) This will send all of the car reports to a file.

8) This will tell the program to not print out any debugging information.

9) Lines 9 - 11 are all comments. I usually leave lines like this in my runfile so that I can switch between options without having to remember what the exact spelling is.

Most of these parameters won't ever need to be changed. Except for GNU_PRINTER, they are all relics from the early days of this project. Instead of taking them out, I have decided to leave them in place in case somebody thinks up a use for them.

12.2 Example 1: Just Car Data

This example will deal only with the car data. Let's say that we had three main directories of car data and the first two had 4 cars worth of data and the last one had 3. So our directory structure inside of the car data directory looked like this:

```
am111492                <= one main directory...
        |
        |-- car1        <= car data...
        |-- car2
        |-- car3
        |-- car4
pm111492                <= main directory...
        |
        |-- car1        <= car data...
        |-- car2
        |-- car3
        |-- car4
am111592                <= main directory...
        |
        |-- car1        <= car data...
        |-- car2
        |-- car3
```

And we wanted to have a lot of reports and we wanted to delete the intermediate files. Then one possible runfile would be:

```
MAIN_DIRECTORY          = am111492 1 2 3 4          (1)
MAIN_DIRECTORY          = pm111492 1 2 3 4          (2)
MAIN_DIRECTORY          = am111592 1 2 3            (3)
REPORT_OPTION           = EVERYTHING                (4)
REPORT_DESTINATION      = FILE                      (5)
CLEAN_UP_OPTION         = DELETE_FILES              (6)
GORE_POINTS             = YES_CALC_GORE_POINTS      (7)
ERROR_FILE_NAME_EXT     =                           (8)
```

Things to note:

- The value of `ERROR_FILE_NAME_EXT` goes to the default value of "err" because nothing was specified.
- The value of `DEBUG_OPTION` goes to the default value because it wasn't even listed.
- When specifying the `MAIN_DIRECTORY` there is at least one space after the directory name and before the numbers of the car subdirectories. If there wasn't then the program wouldn't be able to distinguish between the main directory name and the car subdirectory numbers.

- The gore points are calculated and stored in the appropriate files. The incidents are not calculated and therefore are not stored. The plots of distance vs. time for both the car position and the incident position are saved in appropriate files.

12.3 Example 2: More Car Data

Next, let's say that we had 12 sets of data, some with 4 cars and some with 3, with the names of the various main car directories being:

chevy1	morning1	ford1	ford4
chevy2	morning2	ford2	evening
chevy3	morning3	ford3	day1

So the data is spread out over various directories. Then one possible runfile would be:

```
# This is our runfile. ( This line is a comment line )
MAIN_DIRECTORY = chevy1 1 2 3
MAIN_DIRECTORY = chevy2 2 3 4
MAIN_DIRECTORY = chevy3 1 2 3 4
MAIN_DIRECTORY = ford1 1 2 3 4
MAIN_DIRECTORY = ford2 1 2 3
MAIN_DIRECTORY = ford3 1 2 3
MAIN_DIRECTORY = ford4 1 2 4
MAIN_DIRECTORY = morning1 1 2 3 4
MAIN_DIRECTORY = morning2 1 2 3 4
MAIN_DIRECTORY = morning3 1 2 3 4
MAIN_DIRECTORY = evening 1 4
MAIN_DIRECTORY = day1 1 4
CAR_DIRECTORY_ROOT = car
REPORT_OPTION = EVERYTHING
REPORT_DESTINATION = FILE
CLEAN_UP_OPTION = DELETE_FILES
ERROR_FILE_NAME_EXT =
```

In this example the only output that that will be generated is the diagnostic reports for the cars and those will be stored in the directory defined in the file `fsp_dirs.h` by the defined variable `CARDATA_REPORTS_DIR`.

12.4 Example 3: Lots Of Car Data

This is another car example. In this example we want to generate a couple of different plots for the data in the directory am031093 for car 1. The specifications are as follows:

1. Generate all of the car error reports.
2. Place the car error reports in files.

3. Clean up the temporary car files that were made.
4. Don't spit out any debug information.
5. Include the information from the GORE points.
6. Don't include the INRAD points.
7. Generate the speed-time plots.
8. Don't generate the speed-distance plots.
9. Use the default speed filtering factor.
10. Specify the printer name as "s307".
11. Use the default of everything else.

The runfile for this example follows. Note that you shouldn't put the numbers on each line - those are just there to point out which step is which.

```
# Example runfile
#
REPORT_OPTION           = EVERYTHING            (1)
REPORT_DESTINATION      = FILE                  (2)
CAR_CLEANUP             = DELETE_FILES          (3)
DEBUG_LEVEL             = SILENT_DEBUG          (4)

GORE_POINT_OPTION       = YES_CALC_GORE_POINTS  (5)
INRAD_POINTS            = NO_INRAD_POINTS       (6)
SPEED_TIME_PLOTS        = YES_SPEED_TIME_PLOTS  (7)
SPEED_DIST_PLOTS        = NO_SPEED_DIST_PLOTS   (8)
CAR_SPD_FILTER_FACTOR   =                       (9)
GNU_PRINTER             =  s307                 (10)

MAIN_DIRECTORY = am031093 1
```

This runfile will process the data in the directory am031093 for car number 1.

12.5 Example 4: General Loop Data Example

Loop data is handled the same way as car data except for the parameter strings. If you had loop data on February 4, 1993 and you had data for the detectors 1, 2, 5, 10, and 15, then the line in the runfile would look like this:

```
LOOP_DIRECTORY = lp020493 1 2 5 10 15
```

So a full fledged loop example looks like this:

```
REPORT_OPTION = FILE
REPORT_DESTINATION = EVERYTHING
CAR_CLEANUP = DELETE_FILES
DEBUG_LEVEL = SILENT_DEBUG
ERROR_FILE_NAME_EXT =
NAV_DATA_FILE_NAME =
KEY_DATA_FILE_NAME =
CAR_DIRECTORY_ROOT =

LOOP_TEXT = LOOP_BOTH_REPORTS
LOOP_DIRECTORY = lp030993 1 2 3 4 5 6 7 8 9 10 11 12 13 15 16 17 18 19
```

A few things to note about this example:

- In this example we still had the parameters that deal with the car data, `CAR_CLEANUP`, `CAR_DIRECTORY_ROOT`, etc. This is perfectly all right. Since there is no car data to process these parameters are read in but then nothing is done with them.
- This example will attempt to produce the text and error report for the loop data set.
- Since the time periods for the loop data set are not specified, the default values will be used.
- Since `LOOP_TEXT` was set to `LOOP_BOTH_REPORTS`, you could assume that the user wanted to generate an error report. But since the default for all of the loop tests is **OFF** no tests are run and therefore the error report won't have any results in it. This is probably a mistake and the user should specify some tests to perform.

12.6 Example 5: Complicated Loop Data Example

A more sophisticated loop data example would entail the following:

1. Generate all of the car error reports. This setting will have no effect since we don't specify any car data sets.
2. Place the car error reports in files. This setting will have also no effect since we don't specify any car data sets.
3. Don't spit out any debugging information.
4. Specify the printer name as "s307".
5. Set the value of PPS to mean "counts per output period."
6. For the loop data generate both an error report and a text report.
7. Start the loop report at 6:00am (i.e.: 21600 seconds since midnight the night before).
8. End the loop report at 8:00am (i.e.: 28800 seconds since midnight the night before).

9. For the loop report, print out a value every 60 seconds.

10. For the loop report, generate the speed, occupancy, and count plots.

11. For the error reports, only test if the speed or occupancy is above a certain value. Don't test for anything else.

The runfile for this example follows. Note that you shouldn't put the numbers on each line - those are just there to point out which step is which.

```
# This runfile was made automatically
# by the PC software ftran.
#
REPORT_OPTION = EVERYTHING                        (1)
REPORT_DESTINATION = FILE                         (2)
DEBUG_LEVEL = SILENT_DEBUG                        (3)
GNU_PRINTER = s307                                (4)

OUTPUT_FLOW_AVG_FACTOR = MATCH_OUTPUT_PERIOD      (5)

LOOP_TEXT = LOOP_BOTH_REPORTS                     (6)
LOOP_START_TIME = 21600                           (7)
LOOP_END_TIME = 28800                             (8)
LOOP_OUTPUT_PERIOD = 60                           (9)

LOOP_FLOW_PLOTS = YES_CALC_ALL_FLOW_PLOTS         (10)

LP_SPEED_HIGH_TEST = YES                          (11)
LP_OCC_HIGH_TEST= YES                             (11)
LP_SPEED_HIGH_THRESHOLD_MPH = 90                  (11)
LP_SPEED_HIGH_THRESHOLD_NUM = 2                   (11)
LP_OCC_HIGH_THRESHOLD_PERCENT = 50                (11)
LP_OCC_HIGH_THRESHOLD_NUM 5                       (11)

LOOP_DIRECTORY = lp030993 1 10 16
```

12.7 Example 6: Computing The Delay WRT The Average

One of the more complicated things to do is to calculate the delay with respect to the average. The loop averages are the averages over the all of the days of the speeds, counts, occupancies, and densities. These files are used to calculate the delay for each incident with respect to the average and to generate the contour plots of differential density. The nice thing about the loop averages is that they probably only have to be calculated once. Unfortunately, that first calculation is a bit hard. For programming reasons this calculation needs to be done in a couple of passes of the program. The steps that we are going to follow are given below:

1. In the first pass of the program we will calculate the standard speed, flow, occupancy, and density values for each loop detector for each day. We will also calculate the average of these values over the days.

2. In the second pass we will calculate the delay at each loop detector with respect to the average.

3. As a final step we will copy the delay files to a special directory so that they are not corrupted by further runs of the program. This is done with the **fsp**-generated program called **copydat**.

Since we are only going to be dealing with the loop data in the runfile listings that follow I will not include any parameters that deal with the car data or the incident data.

12.7.1 The First Pass: Standard Values

In this pass we will calculate the speeds, flows, occupancies, and densities for the loop data. We will also calculate the averages for these values over the days. Note that the averages are only generated from the loop data specified. If you only specify 1 day of loop data then you'll have a pretty boring average. If you look below you will notice that we specify that we want to calculate the traffic delay with respect to a constant. Actually, we don't really care about the traffic delay at this point. The reason that we have to run the routine that calculates the traffic delay is because that's where the densities are calculated as well. The density calculation routine probably shouldn't be mixed in with the delay calculation stuff but tradition is a very strong force.

Once the standard values are calculated the program will attempt to calculate the average for these values over all the days. It will place these values in a subdirectory named **Avg** under the main loop output directory. The runfile to do all of this is given below. Note that this runfile is distributed with the source code for the **fsp** project. It's name is `lp.avg.1.run`.

In the list that follows there are quite a few runfile parameters that have been left out because they don't have anything to do with what I am trying to show. You should set those to appropriate values when running the **fsp** program.

```
LOOP_START_TIME          = 18000
LOOP_END_TIME            = 72000
LOOP_OUTPUT_PERIOD       = 300
OUTPUT_FLOW_AVG_FACTOR   = MATCH_OUTPUT_PERIOD
CAR_DATA_SET_NUMBER      = 1
LOOP_DATA_DIRECTORY      = /home/pal2/FSP/Set1/Loopdata
CAR_DATA_DIRECTORY       = /home/pal2/FSP/Set1/Cardata
INCIDENT_DATA_DIRECTORY  = /home/pal2/FSP/Set1/Incidents
OUTPUT_DIRECTORY         = /home/pal2/FSP/Tempout
FLOOP_CLEANUP            = DELETE_EVERYTHING
GLOOP_CLEANUP            = DELETE_NOTHING
HLOOP_CLEANUP            = DELETE_NOTHING
LOOP_FLOW_PLOTS          = YES_CALC_ALL_FLOW_PLOTS        (1)
LOOP_TEXT                = LOOP_ERR_REPORT_ONLY           (2)
```

```
DROPOUT_TIMES            = YES_DROPOUT_FILE              (2)
LOOP_DATA_COMPRESSED     = DATA_IS_COMPRESSED
LOOP_CONSISTENCY_FIX     = NO_FIX_CONSISTENCY_ERRORS
LOOP_HOLES_FIX           = YES_FIX_HOLE_ERRORS           (2)
TRAFFIC_DELAY            = YES_CALC_TRAFFIC_DELAY        (3)
DELAY_CALCULATION        = WRT_CONSTANT_SPEED            (3)
DELAY_TYPE               = ONLY_HAVE_POSITIVE_DELAY      (3)
TRAFFIC_LOW_SPEED        = 55                            (3)
LOOP_AVERAGE             = YES_LOOP_AVERAGE              (4)
LOOP_DIRECTORY = lp021693 1 2 3 4 5 6 7 8 9 10 11 12 13 15 16 17 19 20
LOOP_DIRECTORY = lp021793 1 2 3 4 5 7 8 9 10 11 12 13 15 16 17 19 20
LOOP_DIRECTORY = lp021893 1 2 3 4 5 6 7 8 9 10 11 12 13 15 16 17 18 19 20
```

There are a few things that should be pointed out about the list above. These items are marked on the right hand side with various numbers:

1. The item marked `(1)` tells the **fsp** program to generate the speeds, occupancies, and flows for the loop detectors.

2. The items marked `(2)` are necessary to specify that the program should fix the holes in the loop data. The average probably isn't going to be very good if you don't use the fixed loop data.

3. The items marked `(3)` tell the program to calculate the delay on the loops with respect to a constant. Note that a side effect of doing this is that the program calculates the density values for the loops and this is what we will use later on. Note that at this point we can not calculate the delay with respect to the average and that it must be done with respect to a constant. The reason for this is that the average files don't exist yet (you are in the process of making them right now).

4. Finally, item `(4)` tells the program to calculate the loop averages and to place them in the special directory called `Avg` under the loop output directory. For this runfile the complete path name of this directory would be `/home/pal2/FSP/Tempout/Loopdata/Avg`. The averages in this example will only be over the three days 2/16, 2/17, and 2/18.

12.7.2 The Second Pass: Calculating The Delay

In the second pass we will use the loop average files that were computed during the last pass to calculate the delay with respect to the average. The listing is given below:

```
LOOP_START_TIME          = 18000
LOOP_END_TIME            = 72000
LOOP_OUTPUT_PERIOD       = 300
OUTPUT_FLOW_AVG_FACTOR   = MATCH_OUTPUT_PERIOD
CAR_DATA_SET_NUMBER      = 1
LOOP_DATA_DIRECTORY      = /home/pal2/FSP/Set1/Loopdata
CAR_DATA_DIRECTORY       = /home/pal2/FSP/Set1/Cardata
```

```
INCIDENT_DATA_DIRECTORY  = /home/pal2/FSP/Set1/Incidents
OUTPUT_DIRECTORY         = /home/pal2/FSP/Tempout
FLOOP_CLEANUP            = DELETE_EVERYTHING
GLOOP_CLEANUP            = DELETE_NOTHING
HLOOP_CLEANUP            = DELETE_NOTHING
LOOP_FLOW_PLOTS          = NO_CALC_LOOP_FLOW_PLOTS        (1)
LOOP_TEXT                = LOOP_NO_REPORTS
DROPOUT_TIMES            = NO_DROPOUT_FILES
LOOP_DATA_COMPRESSED     = DATA_IS_COMPRESSED
LOOP_CONSISTENCY_FIX     = NO_FIX_CONSISTENCY_ERRORS
LOOP_HOLES_FIX           = YES_FIX_HOLE_ERRORS            (2)
TRAFFIC_DELAY            = YES_CALC_TRAFFIC_DELAY         (3)
DELAY_CALCULATION        = WRT_AVERAGE_SPEED              (3)
DELAY_TYPE               = ONLY_HAVE_POSITIVE_DELAY
TRAFFIC_LOW_SPEED        = 55
LOOP_AVERAGE             = NO_LOOP_AVERAGE                (4)
LOOP_DIRECTORY = lp021693 1 2 3 4 5 6 7 8 9 10 11 12 13 15 16 17 19 20
LOOP_DIRECTORY = lp021793 1 2 3 4 5 7 8 9 10 11 12 13 15 16 17 19 20
LOOP_DIRECTORY = lp021893 1 2 3 4 5 6 7 8 9 10 11 12 13 15 16 17 18 19 20
```

A couple of things to point out about this listing:

1. We no longer need to generate the standard loop files of speeds, occupancies, and flows because this was done in the 1st pass. This is done on the line labeled `(1)`.

2. We still need to specify that we want to use the loop data that has had the holes fixed. This is done on the line labeled `(2)`.

3. We now calculate the loop delay with respect to the average instead of with respect to a constant. This is done on the lines labeled `(3)`.

4. Finally, note that on the line labeled `(4)` that we don't have to compute the average more than once.

12.7.3 The Final Step: Moving The Files To A Safe Place

Once the delay with respect to the average files have been calculated we need to move them to a special directory so that they will not be overwritten by subsequent passes of the **fsp** program. This special directory is called `Delay` and it is located in the main loop output directory. Underneath the `Delay` directory is a directory for each day of loop data. These directories will hold the loop delay files that were computed with respect to the average.

Note that safety is not the only reason for moving the loop delay files files to this special directory. When the program attempts to calculate the delay for each incident it reads in the loop delay files that correspond to the space-time box that the incident covers and adds up the delay in that box. Well, there are two different places the program can get the delay files from depending on what type of calculation it is doing. If the program is doing the delay calculation with respect to a constant then the loop delay files are pulled out of the normal

loop day directories. The program assumes that the files there were calculated with respect to a constant. If the program is doing the delay calculation with respect to the average then the loop delay files are pulled out of the `Delay` directory structure.

These files can be moved with the help of a shell script program that is generated by the **fsp** program in the loop output directory. If you change into the loop output directory you will see that there is a file called **copydat**. This file will copy all of the loop delay files to the appropriate place under the `Delay` directory. To run this program simply type **copydat** at the command line.

```
clair 10: copydat
Starting to copy files...
 Processing: lp021693
 Processing: lp021793
 Processing: lp021893
Done
```

Once you have copied the files the program will be completely set up to take advantage of the average loop delay files.

12.8 Example 7: Generating The Contour Plots

One very helpful analysis tool is to be able to generate the various contour plots of the loop data with the incidents on them. This is helpful when you want to draw the space-time boxes around the incidents so that you can get a more accurate measurement of the delay per incident. The runfile that is needed to do this is straight forward. The only requirement is that the loop average files need to have been calculated. This was done in Section 12.7.

The steps that we have to take to generate the contour plots are pretty straight forward:

1. We don't have to do any more calculation with the loop data. This, of course, assumes that the loop delays have already been calculated.

2. We need to tell the program to process the incidents and to generate the contour plots with those incidents.

3. We need to create an incident filter to our liking. In the example below we will filter out all of the accidents that occurred within our shift.

Once again, a few parameters have been left out because they don't concern us. This runfile is also in the source code that came with the **fsp** program. It is called `contour.run`:

```
LOOP_FLOW_PLOTS          = NO_CALC_LOOP_FLOW_PLOTS
LOOP_TEXT                = LOOP_NO_REPORTS
DROPOUT_TIMES            = NO_DROPOUT_FILES
LOOP_CONSISTENCY_FIX     = NO_FIX_CONSISTENCY_ERRORS
LOOP_HOLES_FIX           = YES_FIX_HOLE_ERRORS          (1)
```

```
TRAFFIC_DELAY            = NO_CALC_TRAFFIC_DELAY
LOOP_AVERAGE             = NO_LOOP_AVERAGE
LOOP_AGGREGATE_VALUES    = NO_CALC_AGGREGATE_VALUES
LOOP_DIRECTORY = lp021693 1 2 3 4 5 6 7 8 9 10 11 12 13 15 16 17 19 20
LOOP_DIRECTORY = lp021793 1 2 3 4 5 7 8 9 10 11 12 13 15 16 17 19 20
LOOP_DIRECTORY = lp021893 1 2 3 4 5 6 7 8 9 10 11 12 13 15 16 17 18 19 20
PROCESS_INCIDENTS        = YES_PROC_INCIDENTS          (2)
FIX_INC_DURATION         = NO_FIX_INC_DURATION         (3)
FIX_INC_LOCATION         = YES_FIX_INC_LOC             (3)
FIX_INC_DELAY_BOX        = NO_FIX_INC_DELAY            (3)
INC_CONTOUR_DELAY_PLOT   = YES_INC_CONTOUR_DELAY_PLOTS (4)
CORRELATE_CARS_DATABASE  = NO_CORRELATE                (5)
```

Some things to point out about the runfile listed out above:

- All of the loop calculation parameters are turned off. It is assumed that the loop averages have already been made as was done in Section 12.7.
- We still need to specify that the holes in the loop data were fixed as is done in the line labeled `(1)`. Actually, this parameter is used throughout the program to indicate whether we should use the **floop**, **gloop**, or **hloop** files. So it's required that you always specify this.
- We need to tell the program to process the incidents as is done in the line labeled `(2)`.
- The various incident fixes can be on or off, it doesn't matter. You can see in the lines labeled `(3)` that I have chosen to fix the incident locations but not their durations or the delay boxes.
- The most important item is on the line labeled `(4)`. This line tells the program to generate the contour plots. The program will make contour plots for the delay, density and differential density.
- We don't need to correlate the incident database with the car data (actually, we probably never want to do this because it takes too long).
- Note that there are a few parameters that deal with the incident output that were not mentioned.

As was mentioned above, this example is going to generate the contour plots with the accident incidents on them. Therefore, we need to provide an incident filter that will filter out the accidents. The incident filter that was used to do this is given below:

```
DATE                     = 2/16/93 - 2/19/93    (1)
INCIDENT_TYPE_2          = 1 2                  (2)
BEGIN_END                = 0                    (3)
```

12.9 Example 8: Fixing The Incident Locations

One useful thing to be able to do is to fix the incident locations with the help of the probe vehicle data. This is going to be done with the incident database-probe vehicle correlation routine. Much like the example involving the calculation of the loop averages in Section 12.7, this procedure involves a couple of steps. The steps that we will take to do this are as follows:

1. In the first pass of the **fsp** program we will specify the incident and car data that we wish to examine and have the program generate the corresponding correlation plots.
2. Next, we will print these plots out and figure out which incidents should be shifted. We will then code this information into the incident location fix file.
3. In the second pass of the **fsp** program we will tell the program to attempt to fix the incident locations with the incident location fix file that we just made.
4. In the final step we will look at the results of the correlation that we just and examine the new incident location fix table that the **fsp** program generates. This table will then become our final incident fix file.

12.9.1 Step 1: Generating The First Plot

In the example that follows I will only deal with one shift of car data and incidents. In reality, you should probably do this for all of the incidents but it would just take too long in this example. The incident filter that we will use only pulls out the incidents that occurred on February 22, 1993 during the morning shift. It is given below:

```
DATA_TYPE              = F
DATE                   = 2/22/93 - 2/23/93
SHIFT                  = 0
```

The first runfile that we will need should just process the appropriate car data and then generate the correlation graph. A runfile that does this is given below. Note that there are quite a few parameters that aren't provided in the runfile. These were left out because they obscure the message. You should set them to appropriate values when attempting this fix.

```
CAR_DATA_SET_NUMBER     = 1
LOOP_DATA_DIRECTORY     = /home/pal2/FSP/Set1/Loopdata
CAR_DATA_DIRECTORY      = /home/pal2/FSP/Set1/Cardata
INCIDENT_DATA_DIRECTORY = /home/pal2/FSP/Set1/Incidents
OUTPUT_DIRECTORY        = /home/pal2/FSP/Tempout
GORE_POINT_OPTION       = YES_CALC_GORE_POINTS          (1)
INCIDENT_POINTS         = YES_INCIDENT_POINTS           (1)
INRAD_POINTS            = YES_INRAD_POINTS              (1)
SPEED_TIME_PLOTS        = NO_SPEED_TIME_PLOTS           (2)
SPEED_DIST_PLOTS        = NO_SPEED_DIST_PLOTS           (2)
TIME_DIST_PLOTS         = NO_TIME_DIST_PLOTS            (2)
MAIN_DIRECTORY          = am022293 1 3 4                (3)
```

```
PROCESS_INCIDENTS       = YES_PROC_INCIDENTS
FIX_INC_DURATION        = NO_FIX_INC_DURATION          (4)
FIX_INC_LOCATION        = NO_FIX_INC_LOC               (4)
FIX_INC_DELAY_BOX       = NO_FIX_INC_DELAY             (4)
INC_CONTOUR_DELAY_PLOT  = NO_INC_CONTOUR_DELAY_PLOTS
CORRELATE_CARS_DATABASE = YES_CORRELATE                (5)
INC_CORRELATION_GRAPH   = YES_INC_CORR_GRAPHS          (5)
NUMBER_INC_CORR_GRAPHS  = YES_NUMBER_INC_CORR_GRAPHS   (5)
INC_MATCH_ZERO_WIDTH    = NO_MATCH_ZERO_WIDTH_INC      (6)
```

A couple of things to point out about this listing:

- When processing the car data we needed to have the **fsp** program record the key presses that correspond to incidents. In order to make these more accurate we also need to tell the program to record the gore points and the INRAD points. Remember that these points help to locate the incident key press on the freeway. This is all done on the lines labeled `(1)`.
- On the lines labeled `(2)` we tell the program to not generate the various car trajectory plots. We simply don't need them for what we are doing.
- On line `(3)` we specify the car data that we want to look at. For this day we only had three cars in the field.
- On the lines labeled `(4)` we tell the program to not attempt to fix any part of the incidents.
- On the lines labeled `(5)` we tell the program to do the incident database-probe vehicle correlation and to generate the correlation plots.
- Line `(6)` tells the program that it should leave out incidents that were only witnessed once. I have done this simply because it makes the correlation plots look better. The normal user might want to leave those incidents in.

The main result of the **fsp** program run with this runfile is that there is a correlation file that is saved in the car shift directory. For this example that directory is `/home/pal2/FSP/Tempout/Cardata/am022293` and the gnuplot executable to print the file out is called `incidentcor.gp`. The plot is given in Figure 12.1. This plot gives a really good example of why you would want to attempt to fix the incident locations. If we accept that the car key presses mark the correct location of the incidents then it is obvious that incidents 205, 219, 223, and 238 all need to be shifted.

You could also see from the textual output below that things are not so good.

```
Incident database match statistics:
   Total: # incidents, # covered, ratio = 9, 9, 100.0%
   Total: # time entries, # matched, ratio = 57, 22, 38.6%
   Total: # covered incidents, # changed, avg change(ft) = 9, 7,  1715
   Total: # with new loop, ratio = 0,  0.0%

Probe vehicle match statistics:
```

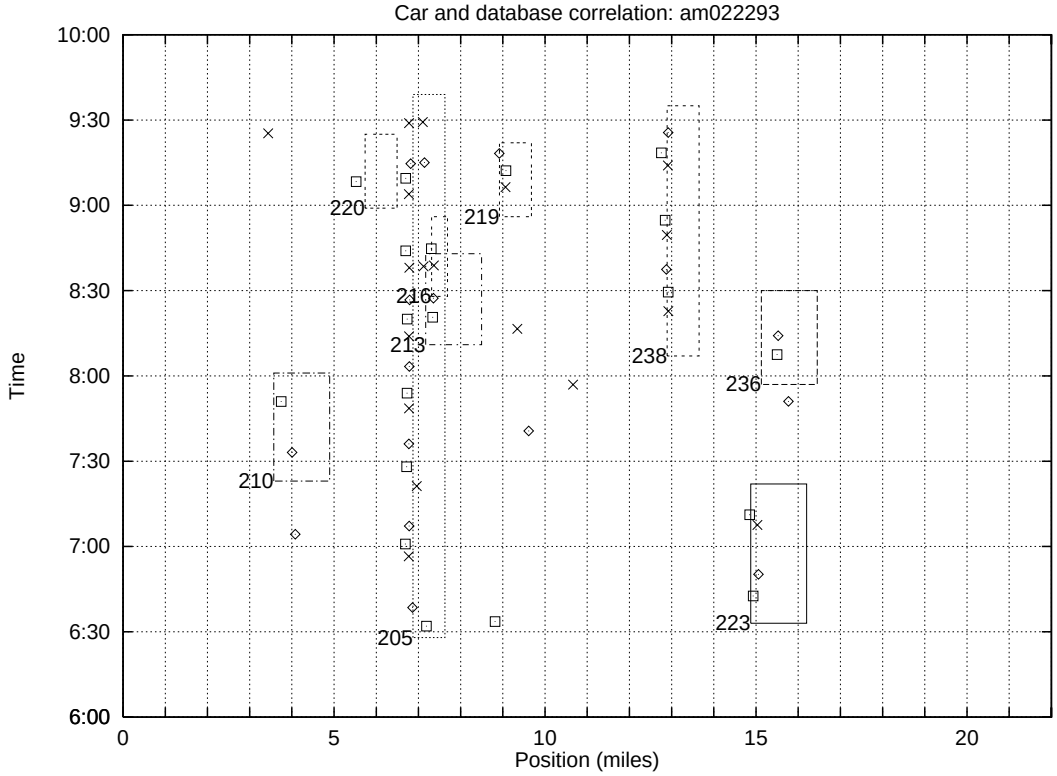

Figure 12.1: Incident Database-Probe Vehicle Correlation Plot.

```
Total number of entries = 102
Number of entries in study section = 54
# matched entries in study section = 22
ratio = 40.7%
```

The statistics above are all interpreted in Section 16.2.3, but what we will look at is the last line which says `ratio = 40.7%`. This means that only 40.7% of the key presses from the probe vehicles were matched with incidents. We will attempt to get this ratio as high as possible.

12.9.2 Step 2: Generating The Location Fix File

From looking at Figure 12.1 you can see how much the incidents need to be moved. Which incidents need to be moved is completely subjective. Take incident 220, for example. There is a key press right before it and a couple right after it. One could argue that this incident should be shifted so that it covers the key press directly to the left. One could also argue that shifting it to the right, to cover the multiple key presses there, is the correct thing to do. Unfortunately, we have not come up with a methodology that will tell us what the drivers actually saw when they pressed their keys. As a result, the shifting of the incidents to cover key presses is completely up to you. I have chosen this day because it is relatively easy to judge what to do for quite a few incidents. If you look at the correlation plots for some of the other days you'll see that in quite a few cases it is nearly impossible to tell how to shift the incidents.

In the input incident directory we will create a file named `inc_fix_loc.dat`. This file will hold the incident numbers that we would like to move and how much we would like to move them. The format is pretty simple: the first column is the incident number and the second column is the amount that we would like to shift the incident (in miles). A positive number means that we would like to shift them to the right on the correlation plot and a negative number means that we would like to shift them to the left. When we are shifting the incidents around we would like to make the incident box land right on top of the key presses. So for incident 223 we should probably shift the incident box 1/2 mile to the left. If we did this the entry in the location fix file would be:

```
223        -0.5
```

A complete incident location fix file for this day would look something like this:

```
205        -0.5
210        -0.25
219        -0.25
220        -0.5
223        -0.5
236        -0.1
238        -0.25
```

Note that there were a few incidents that were not included. For example, incident 213 is buried in a haze of numbers and key presses. Since I couldn't figure out where to shift this incident I didn't shift it at all.

12.9.3 Step 3: Adjusting The Incidents

Once the first incident location fix file has been made we can generate the correlation plot again to see how our fixes performed. This is done by applying almost the same runfile as was used in step 1 except for one change. We need to use the following parameter setting:

```
FIX_INC_LOCATION        = YES_FIX_INC_LOC
```

This, of course, tells the program to read in the incident location fix file[1] that we have just created and use it to shift the incidents around. Note that the shift from the incident location fix routine is done before the correlation routine. Once this has been done we can take a look at the textual report and the correlation plot to see how we did. The correlation plot for this second run is given in Figure 12.2 and the textual output is given below:

```
Incident database match statistics:
   Total: # incidents, # covered, ratio = 9, 9, 100.0%
   Total: # time entries, # matched, ratio = 57, 42, 73.7%
   Total: # covered incidents, # changed, avg change(ft) = 9, 8,   739
   Total: # with new loop, ratio = 5, 55.6%
```

[1]Note that this file *must* be in the incident input directory and it *must* be named "fix_inc_loc.dat."

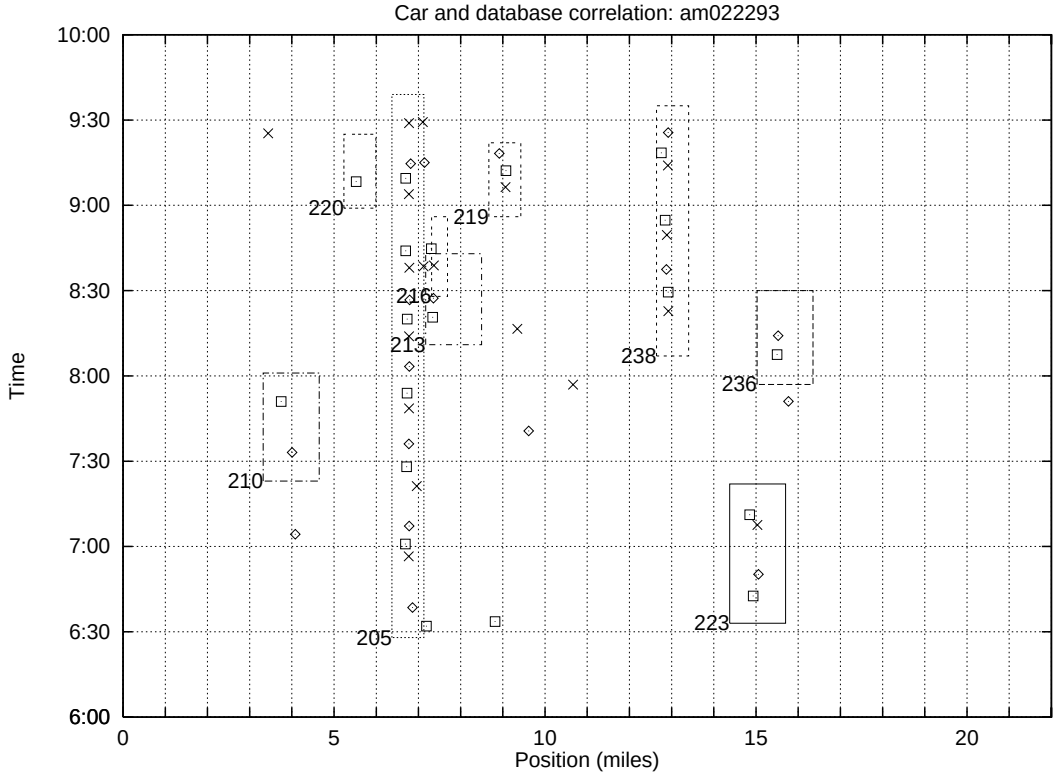

Figure 12.2: Correlation Plot With Fixed Incident Locations.

```
Probe vehicle match statistics:
   Total number of entries = 102
   Number of entries in study section = 54
   # matched entries in study section = 42
   ratio = 77.8%
```

The textual output indicates that 77.8% of the key presses now match up with some incident. From looking at Figure 12.2 we can see that the incidents now match up nicely with the key presses. It's hard to see how we could possibly match up any more. You would think that these results would indicate that we are done, but we aren't. If we were to stop here then each time that we applied the incident location fix the incidents would be shifted only by the amount in the file. The results above say that we only got a 77.8% match, but that was only after the correlation routine shifted the incidents some more. Remember that the correlation routine collects all of the key presses that fall within an incident box and it finds the average location for those key presses. It then sets the new incident location to be this average location. What we want is to make the incident location fix file have the correct shifts such that the correlation routine doesn't have to move the boxes around at all. The way to tell if the correlation routine is moving the boxes around is to look at the table it produces:

```
SUMMARY of all COVERED incidents:
```

Inc #	# Entries	# Matched	Dist(ft)	Loop	New Dist(ft)	New Loop	Change
205	26	19	35640	12	35764	13	0.02
210	3	2	21040	20	20474	7	-0.11
213	3	3	41360	17	38824	17	-0.48
216	2	0	39600	4	39600	4	0.00
219	2	3	47770	5	47619	5	-0.03
220	2	1	29640	18	29160	11	-0.09
223	5	4	79400	1	79031	20	-0.07
236	3	2	82832	1	81901	1	-0.18
238	11	8	68760	19	68003	13	-0.14

This table was generated after the correlation routine was done a second time. The column labeled "Change" tells how much the correlation routine changed the incident location (in miles). This is simply the distance between the column labeled "New Dist(ft)" and the column labeled "Dist(ft)." Note that the column labeled "Dist(ft)" is the location of the incident *after* the incident location fix was applied. Our goal is to make the values in the column labeled "Change" as small as possible.

12.9.4 Step 4: Adjusting One Last Time

The way to make the values in the "Change" column as small as possible is to add the incident shifts that we already placed in the incident location fix file to the values in the column labeled "Change." Fortunately, the **fsp** program will do this for you. It will even generate a new incident location fix table that you can plug directly into the incident location fix file. The table that the **fsp** program generates looks something like this:

```
Possible new incident location fix table:
  Incident    Shift (miles)
     205      -0.48
     210      -0.36
     213      -0.48
     216       0.00
     219      -0.28
     220      -0.59
     223      -0.57
     236      -0.28
     238      -0.39
```

You can now take these numbers and place them in the incident location fix file, `fix_inc_loc.dat` and run the **fsp** program again. The resulting correlation output table will look something like this:

Inc #	# Entries	# Matched	Dist(ft)	Loop	New Dist(ft)	New Loop	Change
205	26	19	35746	12	35764	13	0.00
210	3	2	20460	20	20474	7	0.00
213	3	3	38826	17	38390	4	-0.08

216	2	1	39600	4	38897	4	-0.13
219	2	3	47612	5	47619	5	0.00
220	2	1	29165	18	29160	11	-0.00
223	5	4	79031	1	79031	20	0.00
236	3	2	81882	1	81901	1	0.00
238	11	8	68021	19	68003	13	-0.00

As you can see, the values in the "Change" column have almost all gone to zero. This means that the correlation routine is having almost no effect on the locations of the incidents. In other words, what we have done, by making the incident location fix file, is to subtract off the effect that the correlation routine can have on the incident locations.

There are a couple of things to note about the steps that we just took to produce the final incident location fix file:

- In order for the **fsp** program to generate the new incident location fix table the runfile parameter `FIX_INC_LOCATION` needs to be set to `YES_FIX_INC_DELAY`.

- You might be thinking that you could possibly get all of the numbers in the "Change" column to be zero. Actually, that's pretty hard to do. Sometimes when you shift incidents around they get moved over different key presses and those key presses pull the incident even farther in the direction that it was just moved. The result is some chatter around a stable point. If you get the values to be almost all zero, or even close, then that should be good enough.

- A valid question to ask is why don't we just run the incident correlation routine and take the incident location fix table that the **fsp** program suggests and make that the incident location fix file for future runs. Well, the correlation routine can't figure out if the key presses lie just outside of an incident box or not. It only takes the key presses that fall inside the incident box and it shifts the new location to be the average location of these key presses. This means that you have to get the incidents close to their correct locations before the correlation routine can be useful.

12.10 Example 9: Fixing The Incident Durations

One fix that you can perform on the incident database is the incident duration fix. As was discussed in Section 5.3.2, the incident durations that are recorded in the incident database are not accurate. Since the starting and ending times are only the witnessed times, the durations that are in the incident database are all too short. One way to fix this is to simply add a fixed time to all of the incidents. This is easily done by setting the runfile parameter `HEADWAY_TIME_VAL` to the average probe vehicle headway time. Another, possibly flawed[2], way to correct for the incident durations is to record the times when probe vehicles passed the incident location and didn't witness the incident. You can then make the corrected incident time (either start time or

[2]See the discussion at the end of Section 5.3.2 for an explanation as to why this routine may give biased results.

end time) someplace between the time the incident was witnessed and the time that somebody drove by and it wasn't witnessed.

What we will do here in this example is show how to run the routine that adjusts the incident durations by figuring out when a probe vehicle passed an incident location and the incident wasn't there. This is done in two steps:

- First, we will specify what incidents we wish to examine, determine what probe vehicle data we need, and then tell the **fsp** program to process this data and attempt to fix the incident locations from the probe vehicle data.
- We will then take the incident duration fix file that was generated by the **fsp** program and use that as a runtime file.

12.10.1 Step 1: Using The Probe Data To Correct The Durations

The first thing that we need to do is to attempt to fix the incident durations from the probe vehicle data. To do this we need to specify what incidents we wish to look at and what car data we need. In this example, I will only look at incidents that occurred on February 22, 1993 during the morning shift. An incident filter that will pull these incidents out of the incident database is given below:

```
DATA_TYPE                = F
DATE                     = 2/22/93 - 2/23/93
SHIFT                    = 0
```

The runfile for this example should tell the program to process the probe vehicle data and to try to correct the incident durations using this data. A runfile that will do this is given below. Note that there are quite a few parameters that aren't provided in the runfile. These were left out because they obscure the message. You should set them to appropriate values when attempting this fix.

```
CAR_DATA_SET_NUMBER     = 1
LOOP_DATA_DIRECTORY     = /home/pal2/FSP/Set1/Loopdata
CAR_DATA_DIRECTORY      = /home/pal2/FSP/Set1/Cardata
INCIDENT_DATA_DIRECTORY = /home/pal2/FSP/Set1/Incidents
OUTPUT_DIRECTORY        = /home/pal2/FSP/Tempout
GORE_POINT_OPTION       = YES_CALC_GORE_POINTS          (1)
INCIDENT_POINTS         = YES_INCIDENT_POINTS           (1)
INRAD_POINTS            = YES_INRAD_POINTS              (1)
SPEED_TIME_PLOTS        = NO_SPEED_TIME_PLOTS           (2)
SPEED_DIST_PLOTS        = NO_SPEED_DIST_PLOTS           (2)
TIME_DIST_PLOTS         = YES_TIME_DIST_PLOTS           (3)
MAIN_DIRECTORY          = am022293 1 3 4                (4)
PROCESS_INCIDENTS       = YES_PROC_INCIDENTS            (5)
FIX_INC_DURATION        = FIX_INC_DURATION_FROM_DATA    (6)
INC_DUR_EXPAND_FRACTION = 50                            (6)
FIX_INC_LOCATION        = YES_FIX_INC_LOC               (7)
```

```
CORRELATE_CARS_DATABASE = NO_CORRELATE
INC_MATCH_ZERO_WIDTH    = NO_MATCH_ZERO_WIDTH_INC
INC_FINISHED_OUTPUT     = SCREEN_FINISHED_OUTPUT
INC_FINISHED_OUT_LEVEL  = INC_FIN_OUT_MEDIUM            (8)
```

There are a few things that should be pointed out in the runfile above:

1. When processing the car data we needed to have the **fsp** program record the key presses that correspond to incidents. In order to make these more accurate we also need to tell the program to record the gore points and the INRAD points. Remember that these points help to locate the incident key press on the freeway. This is all done on the lines labeled `(1)`.

2. On the lines labeled `(2)` we tell the program to not generate the speed vs. time and speed vs. distance car trajectory plots. We simply don't need them for what we are doing.

3. But, on line `(3)` well tell the program to generate the time vs. distance plot. When the program generates the time vs. distance plot it also records when the probe vehicles passed the various loop detectors. This information is used when attempting to correct the durations.

4. On line `(4)` we specify the car data that we want to look at. For this example we are just looking at one day.

5. On line `(5)` we tell the program to process the incidents - the incident duration fix routine is in the incident processing section.

6. On the lines labeled `(6)` we tell the program to fix the incident durations from the car data. The expansion fraction is set to 50%. Note that if we hadn't specified any car data that the **fsp** program would not allow us to attempt to fix the incident durations using the car data.

7. On line `(7)` we tell the program to correct the locations of the incidents from the incident location runtime file. We don't have to have this option set - it won't effect the duration fix at all.

8. Finally, on line `(8)` we tell the program to generate a medium amount of diagnostic information about the incident processing. We do this so that the incident duration fix routine will generate a diagnostic table for us.

The output that the **fsp** program generates that we are interested in is in two parts: a diagnostic table that is placed on the screen, and a file that is placed in the incident output directory. The table that was generated by the fix incident duration routine is given below:

```
Incident duration correction statistics:
              Before    New             After    New              Final
Inc    Start   Start   Start    End     End     End   Duration  Duration
 205   23880   23775   23827   34140   34140   34140   10260     10313
 210   27180   26918   27049   28260   28810   28535    1080      1486
```

213	30060	29672	29866	30780	31136	30958	720	1092
216	31080	30440	30760	31560	32663	32111	480	1351
219	32760	31564	32162	33120	33431	33275	360	1113
220	32940	32521	32730	33300	34061	33680	360	950
223	24180	24180	24180	25920	26296	26108	1740	1928
236	29220	28211	28715	30000	30305	30152	780	1437
238	29820	29479	29649	33900	33917	33908	4080	4259

This table lists out the start and end times of each incident as well as the times that a probe vehicle went by the incident location but didn't witness it. The columns labeled "New Start" and "New End" are the new starting and ending times of the incident. These should lie somewhere between the original times and the times when the incident wasn't witness. The column labeled "Duration" is the original incident duration and the column labeled "Final Duration" is the corrected duration.

The second piece of information that is generated by the fix incident duration routine is the file `inc.duration.out` that is placed in the incident output directory. This file holds columns 1, 3 and 6 from the table above. With these columns you can adjust the incident durations no matter what the expansion fraction is.

12.10.2 Step 2: Using The Runtime File To Correct The Durations

Now that we have the fix incident duration file, `inc.duration.out`, we no longer have to use the probe vehicle data to fix the durations. But before we can use this runtime file we need to copy the file, which was generated in the previous step, to a place where the program will recognize it. In our example the file was place in the incident output directory `/home/pal2/FSP/Tempout/Incidents`. The **fsp** program will expect any incident fix runtime files to be in the incident data directory which is `/home/pal2/FSP/Set1/Incidents`. Also, the file needs to be named `inc.duration.in` instead of `inc.duration.out`. To copy the file can execute the following command from the incident output directory:

```
cp inc.duration.out /home/pal2/FSP/Set1/Incidents/inc.duration.in
```

Now when we attempt to fix the incident durations we can tell the program to use the runtime file instead of the probe vehicle data:

```
FIX_INC_DURATION = FIX_INC_DURATION_FROM_FILE
```

This should work exactly the same as using the probe vehicle data but the running time should be a lot faster.

12.11 Example 10: Calculating The Incident Delay With Space-Time Boxes

Probably the most interesting this to do with the incidents is to calculate the delays. In this example we will go through the steps needed to calculate the delay for each incident by using predefined space-time boxes. We will assume that the loop delay files have all been calculated,

and that the contour plots, with the correct incidents on them, have all been generated. These steps were done in Sections 12.7 and 12.8. Since these major steps are out of the way, what we need to do now is the following:

- Look at each contour plot and decide what the bounding box should be for each incident. Code these boxes into the incident delay box file.
- Run the **fsp** program and specify to calculate the incident delay using the bounding boxes.

Note that even though we can't really deal with incidents that overlap, this is probably the best way to calculate the delay for each incident. A complete discussion of the different incident delay calculation can be found in Chapter 11.

12.11.1 Step 1: Figuring Out The Bounding Boxes

The contour plot that I will use in the example is the delay contour for the southbound, morning shift of 2/18/93. I will assume that the delay contour has already been generated. We will make space-time boxes for all of the incidents on this plot. The contour plot is given in Figure 12.3. You can see on this figure that there are a few incidents where the space-time boxes

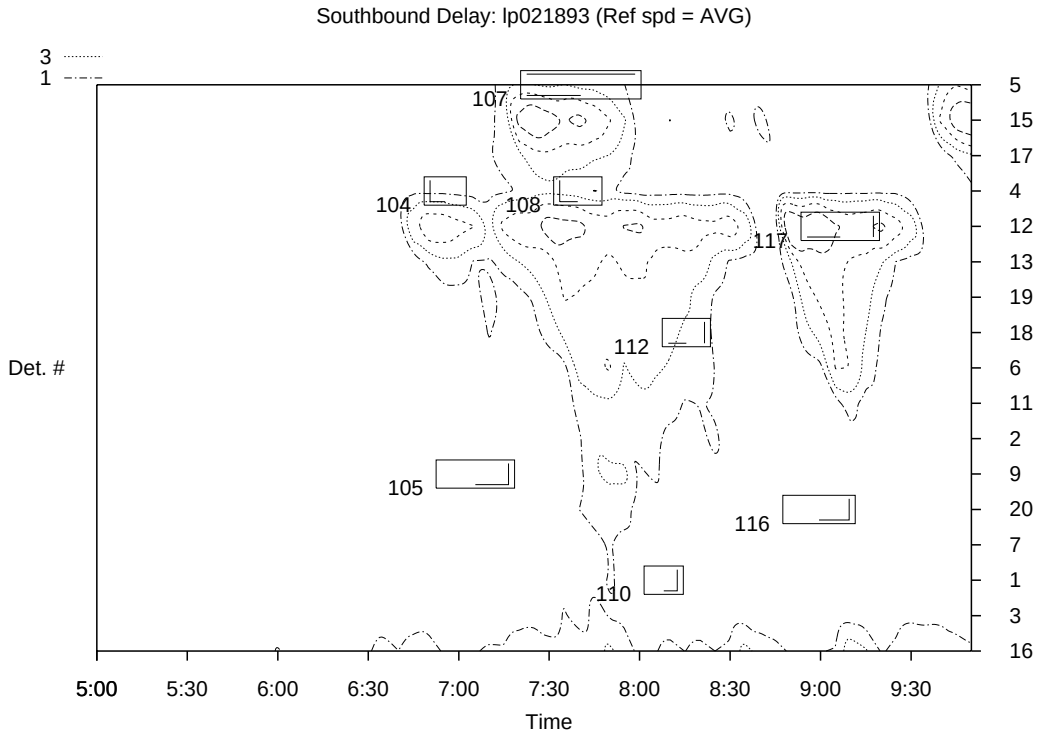

Figure 12.3: Delay Contour Plot.

are really obvious and a few where they aren't. After you have printed this contour plot out you should attempt to to draw reasonable space-time boxes around each incident. For example, for

incident number 117 it is pretty obvious that the whole region around the incident box should be included. On the other hand, you'll need to define the boundary between incidents 104 and 108 pretty carefully.

Code these boxes into an ascii text file with the following 5 columns: incident number, the amount of time, in minutes, that you should extend the box to the left, the number of loop detectors that you should go upstream, the amount of time, in minutes again, that you should extend the box to the right, and finally the number of loop detectors that you should go downstream. So the order of changes that you are describing is left, down, right, up. The number of detectors that you should go upstream includes the current detector. So a value of 1 means that you should only look at this detector, a value of 2 means that you should look at the current detector and the one just upstream, and so on. In some cases where incidents overlapped and there was a small incident that was right in the middle of the delay from another big incident we decided that the delay should be zero. In this case we said that the number of upstream detectors should be 0, which meant that it shouldn't include any detectors. Whether this is the correct thing to do when you have multiple incidents is still up for debate. If you take a look at incident number 112 on Figure 12.3 you can see that this is exactly what happened. In our coding scheme we will make the delay for this incident zero.

The incident space-time boxes that I generated for this contour plot are given below:

```
# Incident  Left  Down  Right    Up
    104      20     4     10     0
    105       0     1      0     0
    107      10     3      0     0
    108      20    11     60     0
    110       0     1      0     0
    112       0     0      0     0
    116       0     1      0     0
    117      10     7     20     1
```

Any line that begins with a "#" sign is a comment line. I put the first line in the file to remind myself what all of the numbers mean. If you look at incident number 104 you'll see that we thought that the bounding box should stretch 20 minutes to the left of the reported incident start time, it should stretch 4 loop detectors upstream, and 10 minutes to the right of the incident end time. For incident number 105 you can see from the coding scheme that we didn't change the times at all and that we specified a bounding box that only included the current loop detector. The reason that we did this is because we looked at the contour delay plots and we noticed that there wasn't any appreciable delay. So as a default we simply specified that the program should only look at the current detector. Note that there has to be spaces between the numbers. Not commas, not hyphens, but spaces.

Finally, this file should be placed in the incident data directory and it should be named `fix_inc_delay.dat`. Note that the file name is specific and if the program doesn't find this file then it will halt. You should also make sure that the program has permission to read this file.

12.11.2 Step 2: Incident Delays From The Bounding Boxes

Now that the incident delay box file is in the proper place we can run the **fsp** program and tell it to calculate the incident delays with the bounding boxes. Of course, the key runfile parameter to change is:

```
FIX_INC_DELAY_BOX = YES_FIX_INC_DELAY
```

A section of the output from this run should look like this:

```
Individual incident statistics:
              Inc. Type                                Good   Bad
Inc #   Date   D 1 2 3   Time South Link Loop  Files Files  Duration     Delay
 104  2/18/93  0 0 2 0   6:52   1    14     4     4     0     0:07:00     38.86
 105  2/18/93  0 5 0 0   6:56   1     7     2     1     0     0:19:00      0.13
 107  2/18/93  0 0 2 0   7:24   1    17     5     3     0     0:33:00    129.55
 108  2/18/93  0 0 2 0   7:35   1    14     4    11     0     0:09:00    445.16
 110  2/18/93  0 5 0 0   8:05   1     3     1     1     0     0:06:00      0.09
 112  2/18/93  0 0 2 0   8:11   1     8    11     8     0     0:09:00      0.00
 116  2/18/93  0 5 0 0   8:51   1     5    20     1     0     0:17:00      0.19
 117  2/18/93  0 0 2 0   8:57   1    14     4     8     0     0:19:00    211.95
```

As you can see, the incidents where we defined large bounding boxes have large delays and the incidents that had small bounding boxes don't. There are a few things to note about calculating the incident delays this way:

- If an incident is not listed in the delay box file then the incident will be discarded. Note that it is not counted as having zero delay, it is thrown away completely. This takes place after the incidents have passed through the incident filter.
- If you are using this method to calculate the delay then you should set the vehicle headway time to be zero. It doesn't make any sense to say that you should add some extra time to the incident durations when you are calculating the delays this way.
- If you use this feature then a few other features become obsolete. These are the incident duration fix, and the incident location fix. Since we are defining the incident delay to be only the delay inside the bounding box the actually incident location and duration are irrelevant. You can leave them on if you want to but they won't effect the delay calculation.
- It would be nice if the computer could draw these bounding boxes by simply looking at the delay contours. This would eliminate the time consuming and irritating step of having to draw the boxes by hand and then code them in. It seems that from the simple picture above that this would be possible. If you look at a typical delay or density contour plot you will quickly realize that this is almost impossible to do (at least by me).

Chapter 13

Program Output: How To View It

Since the Freeway Service Patrol data set contains so many different types of data it is useful to be able to view them graphically. The **fsp** program can be thought of as a big filter that is used to extract various pieces of data from the I-880 data set and to generate plots. The plots that this program generates are specifically designed to be used with **gnuplot** version 3.5, but you can use **xgraph** to display most of the data.

13.1 GNUPLOT

The **gnuplot** program comes with the gnu software that is publicly available. If you don't have it you could download it from gatekeeper.dec.com via anonymous ftp. You could also just tell your system administrator that you need the **gnuplot** program and ask her to get it. Make sure that they get the **gnuplot_x11** program to display the stuff on your screen. It shouldn't be that hard.

The **gnuplot** program needs two files to run. One file is the data file in two columns, x vs. y. This is called the gnuplot data file. The other file is the gnuplot executable file. This is the file that holds all of the information for the plot, like what to use for the title, what range to use for the x and y axis, where to display the plot, etc. My program generates the data files and the executable files automatically. To display a particular set of data then just type:

```
gnuplot gnuplot.executable.filename
```

This will find the data, make the graph and display it on the screen. In certain cases, the **fsp** program will also make gnuplot executable files that will direct the graph to a printer instead of to the screen.

The advantages to using **gnuplot** is that you can make really nice plots. The labels can be perfect, the axis are exactly how you want them, etc. The bad part, is that if you don't have the **gnuplot** executable file then it's hard to get anything. If you want to make plots of anything other than what I planned then you probably shouldn't try to do it with **gnuplot**. The only hard part about using **gnuplot** is that you have to figure out what the gnuplot executable file is named for each type of plot. Since the **fsp** program generates many, many different types of plots this can sometimes be very confusing. The various gnuplot executable files that the program makes are discussed in Chapters 14, 15, and 16. To plot a particular set of data you just need to find the gnuplot executable file that goes with it.

13.2 XGRAPH: An Alternative

The program **xgraph** is a very quick and simple way to generate graphs. It can be used as an alternative to **gnuplot**. It only takes in one filename, which is the data in two columns, and then generates a graph. Once this graph is made you can then dump it to a printer. The problem with **xgraph** is that you can't really set the axis to be whatever you want, the labeling isn't that great, etc. But, if you just want to see the data really fast then this is probably the best way to do it. The really nice thing about **xgraph** is that in Unix you can use shell wildcard characters in specifying the data sets to graph. For example, if you wanted to graph the counts data from the northbound section of loop 3 for all of the lanes then you could type in:

```
xgraph floop3.nc1
xgraph floop3.nc2
xgraph floop3.nc3
xgraph floop3.nc4
xgraph floop3.nc5
```

Each one of these commands will read in the appropriate file and will create a plot on the screen. These file names correspond to the count data for the individual northbound lanes of loop 3. To learn more about the naming convention then see Section 15.3. Once the plots are on the screen there is a button that **xgraph** presents to you that allows you to print each one of these out and compare them. But, since you allowed to use wildcard characters in naming the files, instead of typing the five commands listed out above you could type:

```
xgraph floop3.nc*
```

and the wildcard character "*" will expand to be all of these files. This is exactly the same as typing in:

```
xgraph floop3.nc1 floop3.nc2 floop3.nc3 floop3.nc4 floop3.nc5
```

Either of these last two commands will cause **xgraph** to display all of the data sets on one graph and then generate a key in the upper right hand corner. If you have a color monitor then each one of the lines will be a different color. If you don't have a color monitor then the lines will be various patterns of dots and dashes. This feature is very useful when attempting to compare different sets of data.

The **xgraph** program should come with the release of X that is on your system. Talk to your system administrator about where it is on your system. Of course, both of these programs, **gnuplot** and **xgraph**, require that you have X running on your workstations in order to run them. If you don't then you won't be able to see the plots at all. You should note that the file that **xgraph** takes as an argument is the data file whereas the file that **gnuplot** takes is the special file made by the **fsp** program specifically for **gnuplot**.

13.3 LaTeX Tables

Most of the output that the **fsp** program generates is in the form of a plot or a textual table. In some cases it would be nice to have a better table than just an ASCII printout. In those cases

the **fsp** program generates LaTeX tables that have really nice formatting. LaTeX is a typesetting language that allows the user to create nice postscript documents from text files by putting special commands in the text file. When you have a file that was written with LaTeX commands in it you can either view that file on the screen or print it out to a postscript printer. This section will describe the commands that you need to type to view or print any LaTeX files that the **fsp** program might generate.

There are two steps to processing a LaTeX file:

1. The file first needs to be converted from a LaTeX file to what is called a "dvi" file. A "dvi" file is a file that is device independent. This means that the commands in the file are not specific to any output device. A complete description of why a device independent file is need would take us too far afield. The important thing to note is that once you have the "dvi" file that you can display it on the screen with the program **xdvi**.

2. Once you have the "dvi" if you want to display it on the screen then you can use the command **xdvi**. If you want to print it out then you have to translate it to a postscript file. This is done with a program called **dvi2ps**. You can then pipe this file directly to a printer.

Let's do an example of converting a LaTeX file to a postscript file. The file that we will start off with was generated by the **fsp** program and is named `delay.55.tex`. This file holds various tables of delay values for each time period. An explanation of these tables is given in Section 15.4.1. We start with the file `delay.55.tex` and we want to convert it to a "dvi" file so that we can first view the file on the screen. This is done with the following command:

```
clair 1: latex delay.55.tex
This is TeX, C Version 3.141
(delay.55.tex
LaTeX Version 2.09 <14 January 1991>
(/usr/local/tex/inputs/book.sty
Document Style `book' <24 Nov 89>.
(/usr/local/tex/inputs/bk11.sty)) (/usr/local/tex/inputs/amssymbols.sty)
No file delay.55.aux.
[1] (delay.55.aux) )
Output written on delay.55.dvi (1 page, 2956 bytes).
Transcript written on delay.55.log.
```

In the output above the LaTeX program tells us that it has created a file named `delay.55.dvi`. This file can then be displayed at your local workstation by the command:

```
clair 2: xdvi delay.55.dvi
```

Note that the file extension on all LaTeX files that the **fsp** program generates is "tex" and that the file extension on all "dvi" files is "dvi." Once the "dvi" file has been generated it can be printed to your local printer by converting it to a postscript file and then dumping it to the printer. This can be done with the following command:

```
clair 3: dvips delay.55.dvi | lpr -Pyour_printer_name
 [/usr/tools/lib/ps/tex.ps][1]
```

Where "your_printer_name" is the name of your local printer. Note that on some machines, the program **dvips** will dump the postscript output directly to a printer. In this case, you don't need to pipe the output to a printer as well. If you don't want to print the table out to the printer right away then you can generate a postscript file named `delay.55.ps` from the "dvi" file with the following command:

```
clair 3: dvips delay.55.dvi > delay.55.ps
 [/usr/tools/lib/ps/tex.ps][1]
```

Once you have a postscript file you can print it out to the printer by simply typing:

```
clair 4: lpr -Pyour_printer_name delay.55.ps
```

Although it is not required, it is usually a good idea to make the extension of any postscript files "ps." There are a few things that you should note about generating and printing LaTeX files:

- All of these commands assume that the appropriate programs are in your path. These programs are all pretty common on workstation machines but you might have to contact your system administrator to figure out where the programs are kept.
- Whenever the **fsp** program generates a file of tables there is always one table per output period per direction per page. This means that if the output period is 1 minute and you run the program over the whole time period that there is going to be approximately 1400 pages of output. Well, this will take a long time for LaTeX to process and it will take forever to print out. It is recommend that you only process the tables when the time period of interest is very short or the output period is very long.

Chapter 14

Program Output: The Car Data

The first type of data output that will be discussed is the car data. This chapter will talk about the various graphs and tables that the **fsp** program generates for the car data. Graphs that involve both the car data and the incident data, or the car data and the loop data are described in Chapter 16.

14.1 The Car Textual Output

With all of the different types of data there are two types of output that the **fsp** program generates. The first type of output is text output. This includes error reports, summaries of the data, and estimates of the data validity. The second type of output is graphical. This includes various plots of the data that can be displayed on the screen or printed to a printer. This chapter will discuss the different types of car text files that are generated, their different formats, and where they are placed in the directory structure. A similar discussion for the car plots will take place a little later on in Section 14.2.

The first type of car textual output that I will discuss are the error reports. The generation of the car error reports is governed by the runfile parameter `REPORT_OPTION`. For more information on the runfile parameters see Chapter 7. I will assume that this parameter has been set to `EVERYTHING_NUM`. This will generate four different car error reports: a key one, a huge one, a medium one, and a small one. All of these reports are placed under the car output directory in the directory defined by the variable `CARDATA_REPORTS_DIR`, which is defined in the include file `fsp.h`. This variable is current defined to be "Reports." The various files are named with the following scheme: {key,huge,med,sm}ZZZZZ.err. This means that the first few characters are either **key**, **huge**, **med**, or **sm** corresponding to the key, huge, medium, or small report respectively. The Z's correspond to the 3rd through 7th characters of the runfile that was specified for this run. That might seem a little strange but you need to remember that the whole process is under computer control from the data disks to the final reports and this fits into that scheme very nicely. For an overview of the entire data processing flow see Chapter 17. For example, if the runfile was called `rf09230.run`, then the car error reports would be called:

```
key09230.err
huge09230.err
med09230.err
```

sm09230.err

14.1.1 The Key Error Report

The key error report displays statistics about the keys that the drivers press while driving during their shift. This file is only generated when the data set is specified as the second data set. In Section 4.3 there was a discussion about how we decided that we needed more detailed information from the cars. In that section I described how to tell what kind of car data you are dealing with. You should refer to that section if you don't know what the two types of car data sets are. To understand what the various keys are you should refer to Figure 4.4. The key error file lists out quite a few things:

1. The starting time of each run (or loop).
2. The number of times that the southbound start and end key was pressed per run.
3. The travel time between the two southbound keys per run.
4. The number of times that the northbound start and end key was pressed per run.
5. The travel time between the two northbound keys per run.
6. The number of times the southbound gore key was pressed per run.
7. The travel time between the two southbound gore keys per run.
8. The number of times the northbound gore key was pressed per run.
9. The travel time between the two northbound gore keys per run.
10. A summary of the number of times that the drivers hit the keys correctly for each type of key.
11. The drivers name and the fraction of correctly pressed keys for the whole day.

A short sample of a key error report is given below:

```
*******   Car Data Report:   *******
Number of Data Sets:    1

 3- 9-93   PM
                      #South        #North       #South        #North
 Car  Loop#    Start  Points  TT    Points  TT    Gore    TT    Gore    TT
  1          15:20:21
        1    15:51:04   2   0:14:54   2   0:14:27   2   0:09:56   2   0:11:13
        2    16:23:25   2   0:22:15   2   0:15:05   2   0:05:04   2   0:02:47
        3    16:54:15   2   0:13:15   2   0:16:08   2   0:07:59   1
        4    17:27:41   2   0:12:41   1             0             0
        5    17:59:09   2   0:09:30   1             2   0:07:35   2   0:07:21
                       5/5           3/5           4/5           3/5
        driver = Hisham Noeimi    score = 75%
```

This is the key error report for car number 1 on the afternoon shift of March 9, 1993. The label "TT" stands for "Travel Time." The travel times are for the type of key press directly to the left of the column. For example, the fifth column of data, which is the first column that says "TT", lists the travel times for the southbound points. Since we need to have two key presses of a certain type in order to get a travel time if there aren't two key presses then a travel time is not calculated and there is just a blank. If you look at the travel time for the northbound points then you'll see that in the 4th and 5th loop the driver only pressed the northbound key once. This means that they forgot to press it at the start of the run or at the end of the run. Since we don't have two points, we can't calculate a travel time for these cases. The summary at the bottom is just the total number of times that the driver correctly hit two key presses of a certain type in a single run. The score is the percentage of times that they hit all of the different types of keys correctly. It seems that this driver, Hisham Noeimi, did very well for the first two runs and then got lazy later on in the day.

14.1.2 The Huge Car Error Report

The huge car error report displays statistics on every run that every car makes. It lists the starting time, the run time, the total distance traveled, and even tries to give an estimate as to whether the run was good or not. It should be pointed out that this estimate might not be accurate. The best way to see if a particular run was good or not is to plot out the trajectory as explained in Section 14.2. A short sample of a huge error report is given below:

```
*******   Car Data Report:   *******
Number of Data Sets:    1

 3-10-93   AM
   Car   Loop#   Start time   #Points  Run time    Distance  Suggestion
     1             6:32:55     10820    3:20:20     105.9
           1       6:38:29      2074    0:34:34      19.7    Good run
           2       7:13:04      1679    0:27:59      19.3    Good run
           3       7:41:02      2781    0:46:21      20.1    Good run
           4       8:27:23      1831    0:30:31      18.1    Bad distance
           5       8:57:54      1403    0:23:23      19.2    Good run
           6       9:21:17       717    0:11:57       9.4    Bad distance

     3             6:06:04     12449    3:29:28     109.1
           1       6:08:50      2774    0:46:14      28.5    Bad distance
           2       6:52:58      2245    0:37:25      19.7    Good run
           3       7:30:09      2988    0:49:48      19.3    Good run
           4       8:19:30      1982    0:33:02      19.1    Good run
           5       8:52:32      1571    0:26:11      19.8    Good run
           6       9:22:06       889    0:14:49       2.8    Bad distance
```

This is the huge error report for cars number 1 and 3 for the morning shift of March the 10th. The first line of each car lists out the total run time and the total distance traveled for the whole shift. Each line with a loop# lists out the run time and the distance traveled only for

that particular loop. If you look at car 3, loop 1, you will see that the distance traveled for that loop was 28.5 miles. Since the whole course is only around 19 miles it is a pretty safe bet that there was something wrong with that particular loop. A closer study is definitely warranted. The suggestion that is given in the last column is merely an attempt to determine if the data is correct. Don't take this suggest as the final word on whether or not a particular loop is valid.

14.1.3 The Medium Car Error Report

The medium car error report only lists out the drivers name, the number of loops that they made during the shift, the total distance driven during the shift, and an estimate as to whether the GPS data was any good. To estimate whether the GPS data is any good the program just tries to see if the GPS data falls within some bounding box. If it does then it is labeled as good, and if it doesn't then it is labeled as bad. Clearly a better test could be devised. A sample of a medium error report is listed out below:

```
*******   Car Data Report:   *******
Number of Data Sets:    1

 3-10-93   AM
   Car #   Driver name         #Loops  Start time  Tot distance  GPS sug
     1     Jun Huang              6      6:32:55      105.9     Good data.
     3     Adnan Qadeer           6      6:06:04      109.1     Too noisy.
```

This medium error report is for cars 1 and 3 for the morning shift on March the 10th.

14.1.4 The Small Car Error Report

The small car error report just records if there was data taken at all for the car on the specific day. A sample is listed out below:

```
Quick Data Summary:
                car1   car2   car3   car4   car5
 3-10-93 AM      X              X
```

Note that this doesn't give very much information, but if you just want to know if you got the correct data then it could be helpful.

14.2 The Car Graphical Output

There are quite a few plots that are made from the car data. There are plots that are made for each run, or loop, of each car and then there are plots that are made for all of the cars for a particular shift. I will discuss the two different types of plots separately.

14.2.1 The Graphs For Each Loop

When the car data is collected in the field there is only one file, called `nav.dat`, that contains the location data for all of the runs that car made for the entire shift. The first thing that the **fsp** program does is to parse up this large file into the various runs that the car made. It then takes these single run files and generates various types of plots. The various types of plots generated for each run of each car are listed out below:

1. x-y plots
2. time-distance plots
3. speed-distance plots
4. speed-time plots

When the program generates the individual run files by parses the main `nav.dat` file it names them according to the following scheme: "cXloopY.*" Where "X" is the car number, "Y" is the loop, or run, number, and "*" is the file type extension. The file prefix "cXloopY" will be referred to as the base file name. For example, if we were talking about the second loop that car 1 took then the base file name would be "c1loop2" and the base file name for the third loop of car 5 is "c5loop3."

These files are all stored in the individual car directories. For example, the input car directory structure will look something like this:

```
am110492            <= This is the main car directory
     |
     |-- car1       <= Sub car dir 1
          |- fsp.dat
          |- nav.dat
          |- key.dat
          |- gps.dat
     |
     |-- car2       <= Sub car dir 2
     |
     |-- car3       <= Sub car dir 3
                .
                .
```

After processing the output car directory would look like this:

```
am110492            <= This is the main car directory
     |
     |-- car1       <= Sub car dir 1
          |- c1loop1.cxy  <= New file
          |- c1loop2.cxy  <= New file
          |- c1loop3.cxy  <= New file
```

```
                .
                .
|
|-- car2        <= Sub car dir 2
     |- c2loop1.cxy  <= New file
     |- c2loop2.cxy  <= New file
     |- c2loop3.cxy  <= New file
                .
                .
|
|-- car3        <= Sub car dir 3
                .
                .
```

For each type of graph there are five different files. There is the car trajectory, the incident locations, the INRAD locations, the gnuplot executable to view the plot on the screen, and the gnuplot executable to print the plot to the printer. Before I discuss the various file types I would like to explain how the file extensions are created. The base file name, as discussed above, is always going to be of the form "cXloopY." Well, the file extension is three letters long and it is of the form "WZZ." Where "W" signifies the data type and "ZZ" signifies the plot type. The various possible values are listed out below.

1. Data types:
   (a) Car trajectory: "c"
   (b) Incident locations: "i"
   (c) INRAD locations: "r"
   (d) Gnuplot executable to view plot: "v"
   (e) Gnuplot executable to print plot: "p"
2. Plot types:
   (a) X-Y plot: "xy"
   (b) Time-distance plot: "td"
   (c) Speed-distance plot: "sd"
   (d) Speed-time plot: "st"

For example, the file that holds the car trajectory on the speed vs. distance plot would have an extension of "csd." Or the gnuplot plot executable file that would generate a graph on the screen of the time vs. distance plot would have a file extension of "vtd." If we combine this with what we learned above about the base file names then we can form the whole file name. So the file that holds the incident locations for the speed vs. time plot for the second run of the the third car is named "c3loop2.ist."

Figure 14.1 is a quick reference to help you figure out the file extensions based on the plot type and the data type. The two holes that occur in the INRAD row are there because

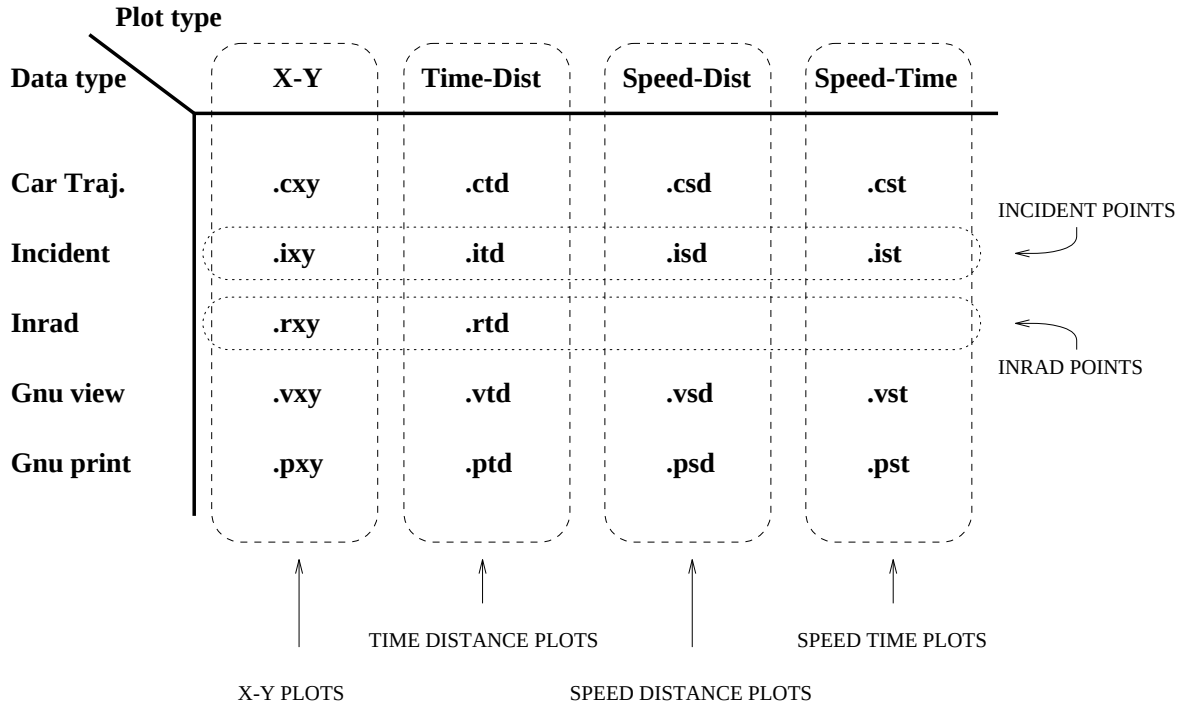

Figure 14.1: Car File Name Extensions.

the INRAD points aren't plotted on the speed vs. distance or speed vs. time graphs, so the files aren't generated by the **fsp** program.

An explanation of the various graph types and the files that go with them follows.

X-Y plots: These plots are the plots of the car trajectory on an X-Y graph. They also have the INRAD points and the incidents marked on the graph.

File type	Extension/File name
Car trajectory on X-Y plot	.cxy
INRAD points on X-Y plot	.rxy
Incidents on X-Y plot	.ixy
gnuplot file to view plot	.vxy
gnuplot file to print plot	.pxy
File to view all X-Y plots for car	viewxy
File to print all X-Y plots for car	printxy

An example of this naming scheme for car 1, loop 1 would be:

```
c1loop1.cxy     <= Data file: car position on X-Y plot
c1loop1.rxy     <= Data file: INRAD points on X-Y plot
c1loop1.ixy     <= Data file: Incidents on X-Y plot
c1loop1.vxy     <= gnuplot file: Executable to view plot
c1loop1.pxy     <= gnuplot file: Executable to print plot
viewxy          <= Script file: View all of the X-Y plots
printxy         <= Script file: Print all of the X-Y plots
```

Time-distance plots: These plots are the plots of the car trajectory on a time vs. distance graph. They also have the INRAD points and the incidents marked on the graph. I have used the abbreviation T-D for time vs. distance below.

File type	Extension/File name
Car trajectory on T-D plot	.ctd
INRAD points on T-D plot	.rtd
Incidents on T-D plot	.itd
gnuplot file to view plot	.vtd
gnuplot file to print plot	.ptd
File to view all T-D plots for car	viewdt
File to print all T-D plots for car	printdt

An example of this naming scheme for car 1, loop 1 would be:

```
c1loop1.ctd     <= Data file: car position on T-D plot
c1loop1.rtd     <= Data file: INRAD points on T-D plot
c1loop1.itd     <= Data file: Incidents on T-D plot
c1loop1.vtd     <= gnuplot file: Executable to view plot
c1loop1.ptd     <= gnuplot file: Executable to print plot
viewtd          <= Script file: View all of the T-D plots
printtd         <= Script file: Print all of the T-D plots
```

Speed-distance plots: These plots are the plots of the car trajectory on a speed vs. distance graph. They also have the incidents marked on the graph. I have used the abbreviation S-D for speed vs. distance below.

File type	Extension/File name
Car trajectory on S-D plot	.csd
Incidents on S-D plot	.isd
gnuplot file to view plot	.vsd
gnuplot file to print plot	.psd
File to view all S-D plots for car	viewsd
File to print all S-D plots for car	printsd

An example of this naming scheme for car 1, loop 1 would be:

```
c1loop1.csd     <= Data file: car position on S-D plot
c1loop1.isd     <= Data file: Incidents on S-D plot
c1loop1.vsd     <= gnuplot file: Executable to view plot
c1loop1.psd     <= gnuplot file: Executable to print plot
viewsd          <= Script file: View all of the S-D plots
printsd         <= Script file: Print all of the S-D plots
```

Speed-time plots: These plots are the plots of the car trajectory on an speed vs. time graph. They also have the INRAD points and the incidents marked on the graph. I have used the abbreviation S-T for speed vs. time below.

File type	Extension/File name
Car trajectory on S-T plot	.cst
Incidents on S-T plot	.ist
gnuplot file to view plot	.vst
gnuplot file to print plot	.pst
File to view all S-T plots for car	viewst
File to print all S-T plots for car	printst

An example of this naming scheme for car 1, loop 1 would be:

```
c1loop1.cst     <= Data file: car position on S-T plot
c1loop1.ist     <= Data file: Incidents on S-T plot
c1loop1.vst     <= gnuplot file: Executable to view plot
c1loop1.pst     <= gnuplot file: Executable to print plot
viewst          <= Script file: View all of the S-T plots
printst         <= Script file: Print all of the S-T plots
```

The files that start with "print" or "view" are script files that allow the user to view or print all of a certain file type right in a row. For example, if the user were to type "printxy" then that would print the x-y plots for all of the loops for that car to the printer without prompting for a return before each file. If there were five loops then this would be the same as typing:

```
gnuplot c1loop1.pxy
gnuplot c1loop2.pxy
gnuplot c1loop3.pxy
gnuplot c1loop4.pxy
gnuplot c1loop5.pxy
```

If you look at the file "printxy", you'll see that that's exactly what it does. The "viewxy" file would allow you to view all of the x-y plots right in a row but it will prompt you between each one for a carriage return. You will notice that there are basically two types of files in each category: the data files and the gnuplot executable files. You should note that you can always print the data files with something like **xgraph** instead of using **gnuplot**. Of course, you won't get the fancy labeling, but you can generate plots very quickly this way. Examples of all of the car plots are given in Section 14.3.

14.2.2 The Graphs For Each Shift

The second category of car plots that the **fsp** program generates deal with the travel times as the cars travel northbound and southbound. The program will generate four different types of plots:

1. Travel times from INRAD points
2. Travel times from northbound gore points
3. Travel times from southbound gore points
4. Travel times from southbound INRAD and southbound gore points

These plots are for all of the cars for a specific shift. These files are all stored in the main car output directory. For example, the final directory structure after the **fsp** program was run would look something like this:

```
am110492            <= This is the main car directory
     |
     |-- car1       <= Sub car dir 1
     |
     |-- car2       <= Sub car dir 2
     |
     |-- car3       <= Sub car dir 3
     |
     |-- car4       <= Sub car dir 4
     |
     |-- inrad.dat  <= New file
     |-- inrad.gtp  <= New file
     |-- inrad.gtv  <= New file
            .
            .
            .
```

The actual starting and ending points that are used to calculate the travel times are either the gore points or the INRAD points. Unfortunately, there is only one INRAD point on the northbound run so we couldn't calculate any travel times for the northbound section based on the INRAD data. There aren't nearly as many travel time plots as there are individual car loop files and therefore the naming scheme is pretty straight forward. An explanation of the various file types follows.

Travel times: INRAD: These are the travel times as computed from the INRAD points for the southbound run. The total distance traveled is about 5.5 miles.

```
inrad.dat <= Data file: INRAD data points
inrad.gtv <= gnuplot file: Executable to view plot
inrad.gtp <= gnuplot file: Executable to print plot
```

Travel times: Northbound gore points: These are the travel times as computed from the gore points for the northbound run. The total distance traveled is about 7.0 miles.

```
ngore.dat <= Data file: gore data points
ngore.gtv <= gnuplot file: Executable to view plot
ngore.gtp <= gnuplot file: Executable to print plot
```

Travel times: Southbound gore points: These are the travel times as computed from the gore points for the southbound run. The total distance traveled is about 7.0 miles.

```
sgore.dat <= Data file: gore data points
sgore.gtv <= gnuplot file: Executable to view plot
sgore.gtp <= gnuplot file: Executable to print plot
```

Travel times: Southbound INRAD and gore points: These are the travel times as computed from the INRAD points and the gore points for the southbound run. This just uses the other data files to generate the plot - it doesn't need any of its own.

```
stimes.gtv <= gnuplot file: Executable to view plot
stimes.gtp <= gnuplot file: Executable to print plot
```

14.3 The Car Plots

Figures 14.2 thru 14.9 are the various plots that can be made from the car data. The gnuplot executable file that was used to view each plot is listed in the caption.

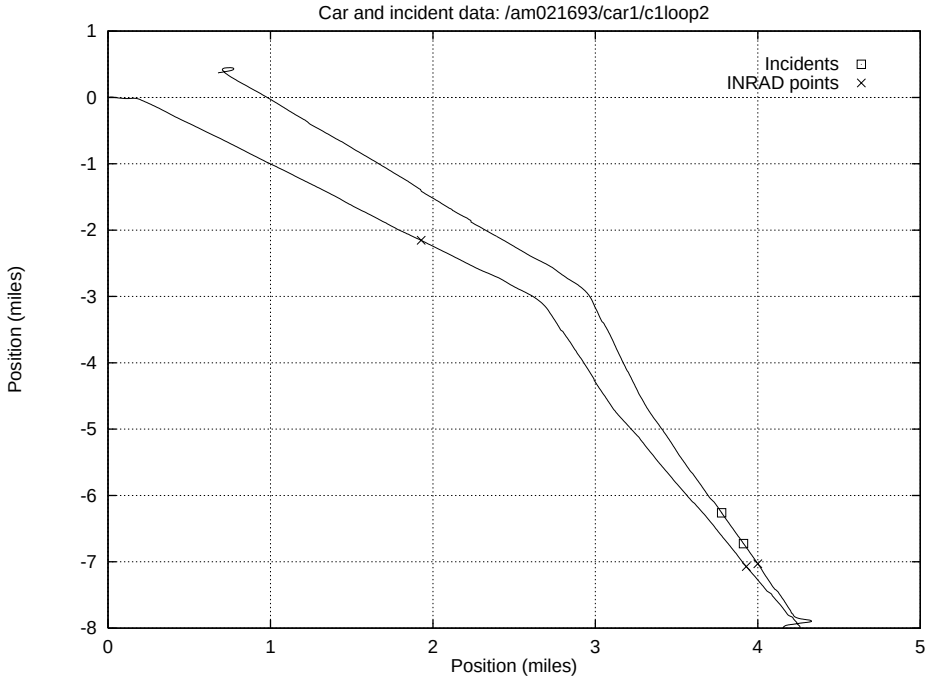

Figure 14.2: Car Trajectory (X-Y). Gnuplot file: c1loop2.vxy

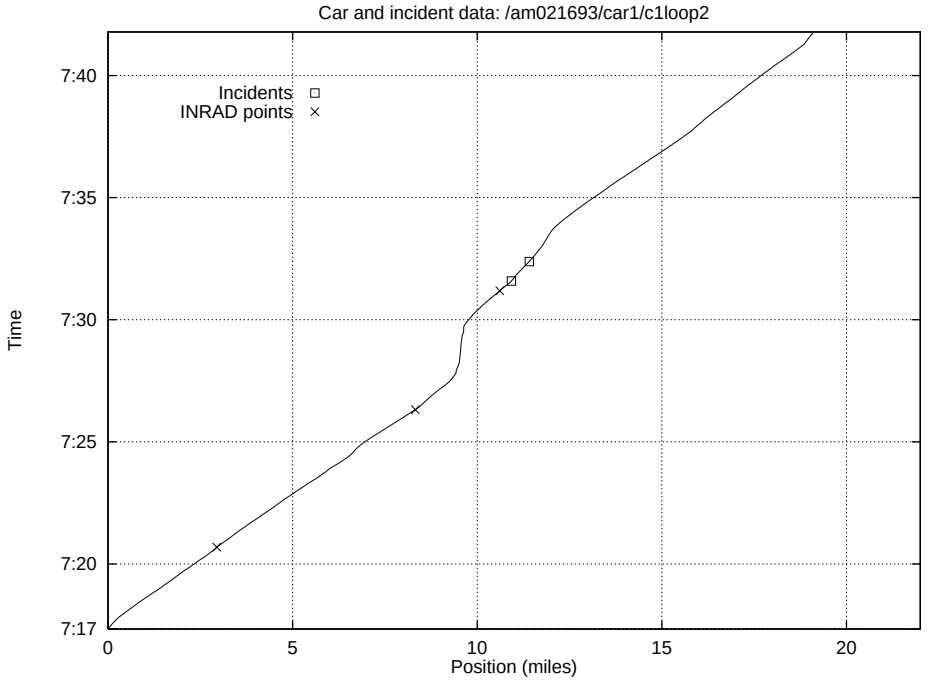

Figure 14.3: Car Trajectory (time vs. distance). Gnuplot file: c1loop2.vtd

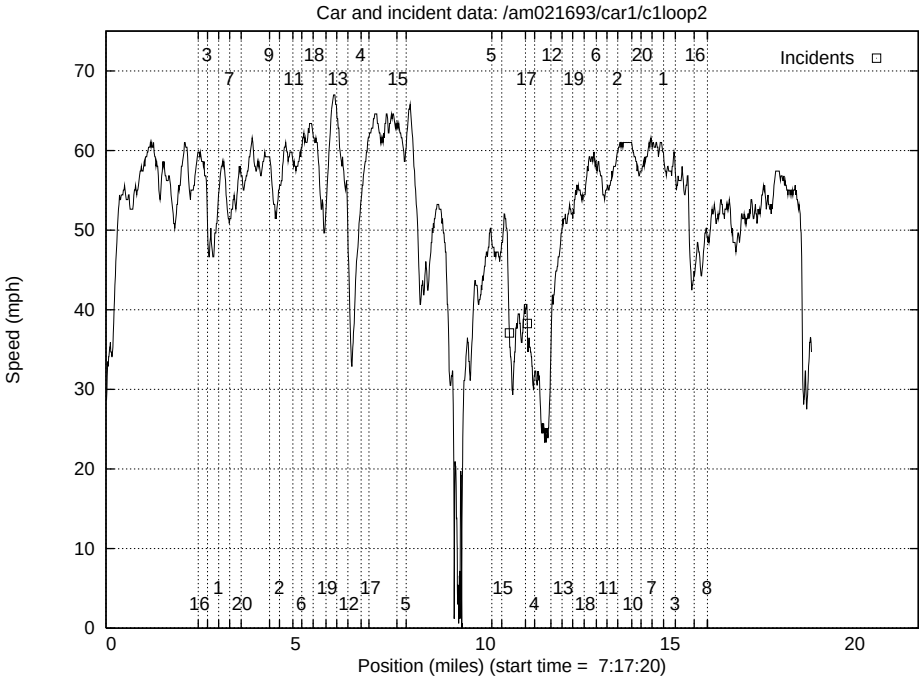

Figure 14.4: Car Trajectory (speed vs. distance). Gnuplot file: c1loop2.vsd

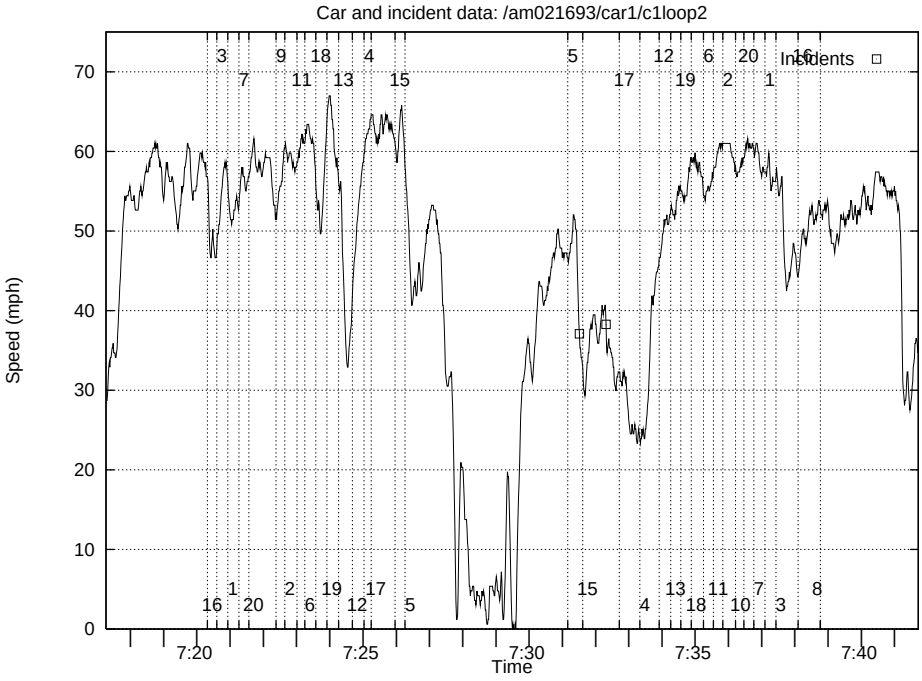

Figure 14.5: Car Trajectory (speed vs. time). Gnuplot file: c1loop2.vst

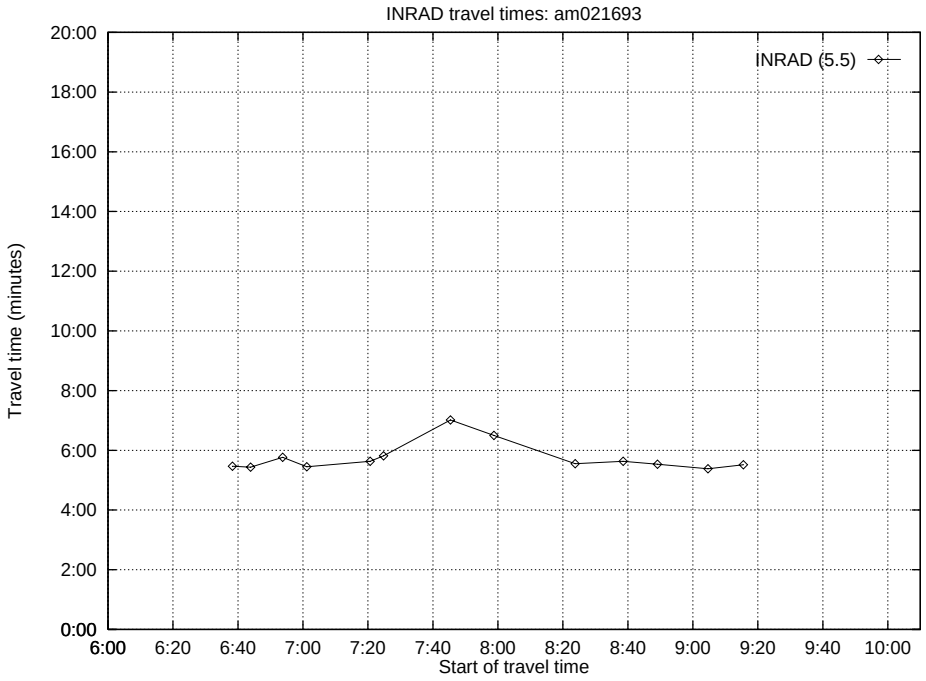

Figure 14.6: Travel Times With INRAD Points. Gnuplot file: inrad.gtv

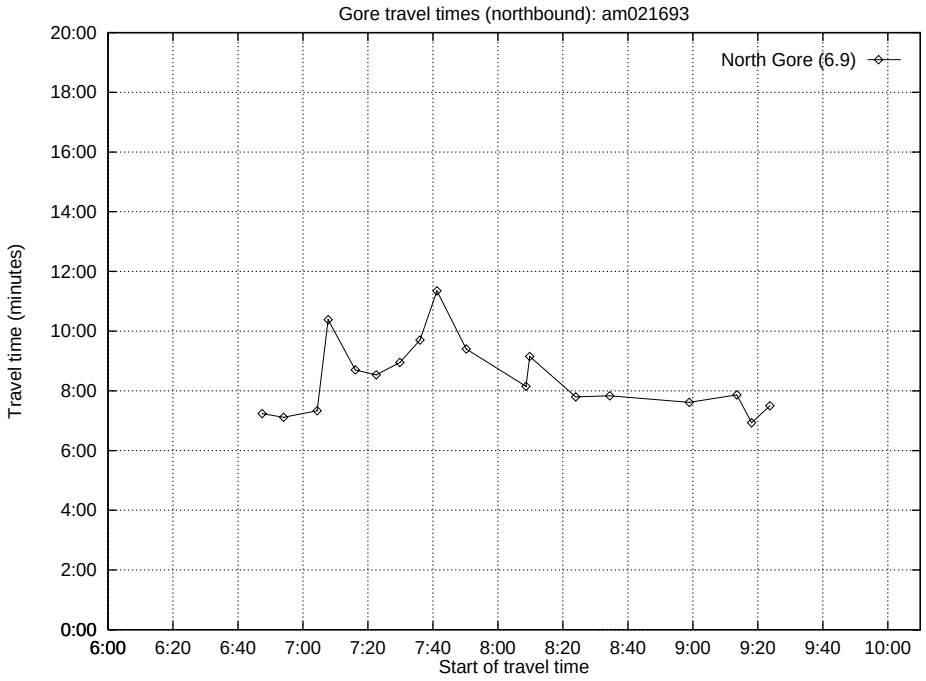

Figure 14.7: Travel Times With nbd Gore Points. Gnuplot file: ngore.gtv

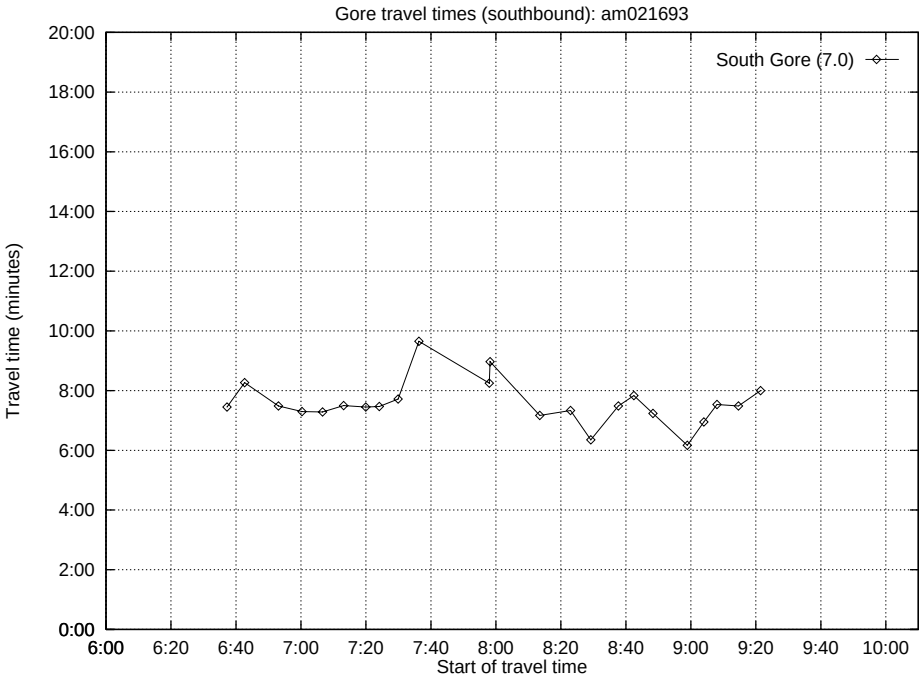

Figure 14.8: Travel Times With sbd Gore Points. Gnuplot file: sgore.gtv

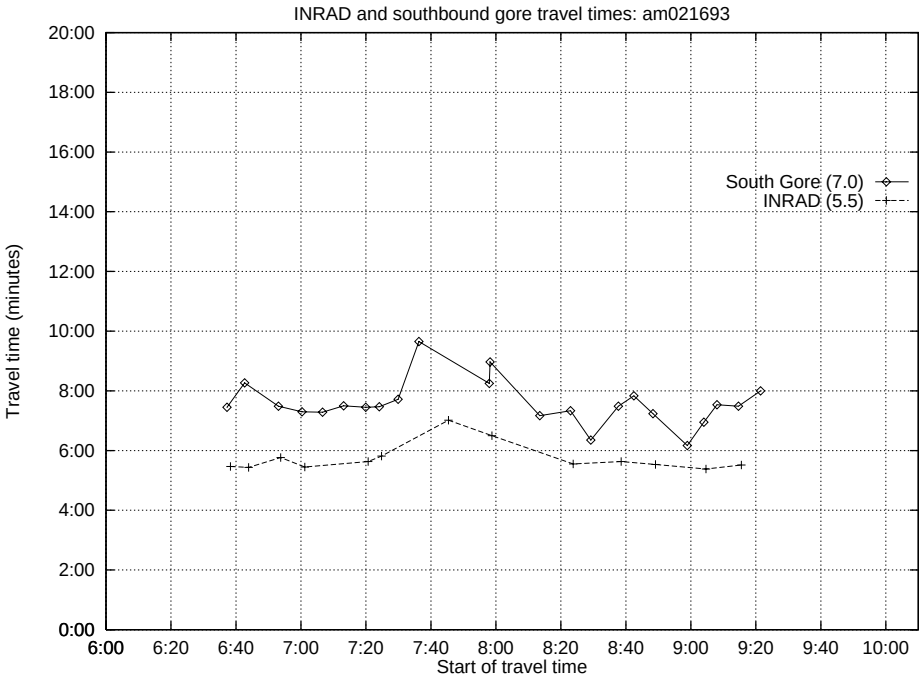

Figure 14.9: Travel Times With Gore And INRAD Points. Gnuplot file: stimes.gtv

Chapter 15

Program Output: The Loop Data

Like the car data there are various types of output for the loop data. This chapter will discuss the textual and graphical output for just the loop data. The discussion of the output that results from the mixture of the loop data with either the car data or the incident database is deferred until Chapter 16.

15.1 The Loop Textual Output

There are three different kinds of text output from the loop data:

1. Text reports for the loop data set.
2. Error reports for the loop data set.
3. Dropout times for the loop data set.

A more thorough discussion of the loop error reports is given in Chapter 10. Below is a brief description of each type of output file.

Text reports for the long data set: These files hold the actual text of the loop data for the long data set. The format of the text records is described in Chapter 4. These files are named in the following scheme: loopZZ.txt, where "ZZ" is the loop number. So if you wanted to find the text report for the long data set for loop number 2 then the file would be named `loop2.txt`. The file extension is defined in the file `fsp_ext.h` as follows:

```
#define         LOOP_TEXT_FILE_EXTENSION        "txt"
```

This can be changed to whatever you prefer. Whether these files are generated or not is determined by the runfile parameter `LOOP_LONG_TEXT`. It needs to be set to either:

```
LOOP_LONG_TEXT_REPORT_ONLY
or
LOOP_LONG_BOTH_REPORTS
```

in order for these files to be made. See the discussion in Chapter 7 for a detailed description of the runfile parameters. You should note that these files take up a lot of space and you should only generate them as a last resort. These files are stored under the loop output directory in the individual day directories.

Error reports for the long data set: These files hold the error reports from the loop data for the long data set. The format of each line in the error report is described in Chapter 10. These files are named in the following scheme: loopZZ.err, where "ZZ" is the loop number. So if you wanted to find the error report for the long data set for loop number 15 then the file would be named `loop15.err`. The file extension is defined in the file `fsp_ext.h` as follows:

```
#define         LOOP_ERROR_FILE_EXTENSION       "err"
```

This can be changed to whatever you prefer. Whether these files are generated or not is determined by the runfile parameter `LOOP_LONG_TEXT`. It needs to be set to either:

```
LOOP_LONG_ERR_REPORT_ONLY
or
LOOP_LONG_BOTH_REPORTS
```

in order for these files to be made. See the discussion in Chapter 7 for a detailed description of the runfile parameters. These files are stored under the loop output directory in the individual day directories.

Dropout times for the long data set: This file holds the list of times that the cabinets were not taking data. Sometimes the cabinets skip an output period or two and sometimes they stop working for hours at a time. What this file holds is a list of the times that there is no data at all from the cabinets. A short segment of one of these files is given below:

```
Summary of Loop Data Dropout Times:
The requested data times are:  5:00 - 10:00, 14:00 - 20:00

/home/clair0/PATH/FSP/Set1/Loopdata/lp031893/loop1:
/home/clair0/PATH/FSP/Set1/Loopdata/lp031893/loop2:
 15:47 - 15:49
 17:19 - 17:21
 17:30 - 17:32
/home/clair0/PATH/FSP/Set1/Loopdata/lp031893/loop3:
/home/clair0/PATH/FSP/Set1/Loopdata/lp031893/loop4:
/home/clair0/PATH/FSP/Set1/Loopdata/lp031893/loop5:
/home/clair0/PATH/FSP/Set1/Loopdata/lp031893/loop6:
  8:29 -  9:45
/home/clair0/PATH/FSP/Set1/Loopdata/lp031893/loop7:
  8:56 -  8:58
```

```
/home/clair0/PATH/FSP/Set1/Loopdata/lp031893/loop8:
/home/clair0/PATH/FSP/Set1/Loopdata/lp031893/loop9:
/home/clair0/PATH/FSP/Set1/Loopdata/lp031893/loop10:
```

The file lists at the top the times that the file covers. In this case the times are from 5:00am until 10:00am and then from 2:00pm until 8:00pm - the whole data set. The program places on each line the filename for each cabinet and then places on the lines below it the times that that cabinet didn't have any data. So cabinet #1 is the first one and there are no lines below it with times. This means that there weren't any times when cabinet #1 didn't have data. But if we look at cabinet #2, it seems that it didn't have data for three different time periods. Something very important to note is that the data is checked only for every output period. So if the output period is set for 15 minutes then there might be a 10 minute segment in the middle of the period where the cabinet isn't taking data and this program wouldn't spot it. To avoid this you should make the output period as fine as possible. The trade off, of course, is speed: the smaller the output period the longer the running time. The output period is set by the runfile parameter `LOOP_LONG_OUTPUT_PERIOD`. Refer to the discussion in Chapter 7 for more details.

The dropout times file for the long data set is named `long.bad.times`. The filename is defined in the file `fsp_ext.h` as follows:

```
#define         LONG_DROPOUT_TIMES_FILENAME     "long.bad.times"
```

This can be changed to whatever you prefer. Whether these files are generated or not is determined by the runfile parameter `DROPOUT_TIMES`. It needs to be set to either:

```
LONG_DROPOUT_FILE
or
BOTH_DROPOUT_FILES
```

in order for this file to be made. See the discussion in Chapter 7 for a detailed description of the runfile parameters. This file is stored under the loop output directory in the individual day directories.

Text reports for the short data set: These files hold the actual text of the loop data for the short data set. They are identical to the files that hold the text for the long data set except that they hold the data for the short set and the file extension is different. These files are named: loopZZ.stxt. The file extension is defined in the file `fsp_ext.h` as follows:

```
#define         LOOP_SUM_TEXT_FILE_EXTENSION    "stxt"
```

This can be changed to whatever you prefer. Whether these files are generated or not is determined by the runfile parameter `LOOP_TEXT`. It needs to be set to either:

```
LOOP_TEXT_REPORT_ONLY
or
LOOP_BOTH_REPORTS
```

in order for these files to be made. See the discussion in Chapter 7 for a detailed description of the runfile parameters. These files are stored under the loop output directory in the individual day directories.

Error reports for the short data set: These files hold the error reports from the loop data for the short data set. These files are almost the same as the long error report files. The difference is that these files hold all of the error reports for all of the loops from a single day and the data dropout report for that day as well. All of these reports are just stored one after the other with the data dropout report, if it is generated, being last. These files are named in the following scheme: XXXXXX.sum. Where "XXXXXX" is the name of the loop directory. So the error reports for the short data set are stored in terms of days instead of in terms of cabinets. If you wanted to find the error report for the short data set for March 10th then the file would be named, according to the naming scheme that I have on my machine, `lp031093.err`. The file extension is defined in the file `fsp_ext.h` as follows:

```
#define         LOOP_SUMMARY_FILE_EXTENSION     "sum"
```

This can be changed to whatever you prefer. Whether these files are generated or not is determined by the runfile parameter `LOOP_TEXT`. It needs to be set to either:

```
LOOP_ERR_REPORT_ONLY
or
LOOP_BOTH_REPORTS
```

in order for these files to be made. The data dropout report will only be placed at the end of the file if the runfile variable `DROPOUT_TIMES` is set appropriately. See the discussion in Chapter 7 for a detailed description of the runfile parameters. These files are stored in the loop output reports directory defined by the variable `LOOPDATA_SUMMARY_DIR` in the file `fsp.h`. This is currently defined to be `Reports`.

Dropout times for the short data set: This file holds the times when the data is not available from the different cabinets. This is basically the same as the long dropout report except that it just covers the short data set. The dropout times for the short data set are placed at the end of the short error report and in the file named `short.bad.times`. This filename is defined in the file `fsp_ext.h` as follows:

```
#define         DROPOUT_TIMES_FILENAME     "bad.times"
```

This can be changed to whatever you prefer. Whether these files are generated or not is determined by the runfile parameter `DROPOUT_TIMES`. It needs to be set to:

YES_DROPOUT_FILE

in order for this file to be made. See the discussion in Chapter 7 for a detailed description of the runfile parameters.

15.2 The Loop Text Reports Summary

Table 15.1 is a short summary of the key points about each type of output text file.

File Type	Runfile Parameter	Default Extension	Output Place
Loop Text	LOOP_TEXT	.stxt	Loop directory
Loop Error	LOOP_TEXT	.sum	LOOPDATA_SUMMARY_DIR
Loop Dropout	DROPOUT_TIMES	bad.times	Loop directory, and error file

Table 15.1: Summary of loop output text files.

15.3 The Basic Data Set

The loop data files that are generated from the raw loop data can be divided into two categories: the basic data set and the calculated data set. The basic data set contains values that were already in the raw data, and the calculated data set contains values that were calculated from the basic data set. Unfortunately, it can get a little more complicated than that. By manipulating the runfile properly you can apply different fixes to the loop data and each loop fix generates a different set of filenames. These complications will be discussed at the appropriate places below.

The basic data set consists of the flows, speeds, and occupancies for each individual loop detector. This data is extracted into an individual file for each value and for each lane at every loop detector. The file has two columns: the first column is the number of seconds since midnight and the second column is the value for that time period. There are also gnuplot executable files that are made to display or print each value. The basic data filename looks something like this:

{f,g,h}loopXX.{n,s}{s,c,o}Y

This is basically a mess when you first look at it. The notation {f,g,h} means that there is either an "f," "g," or an "h" at that one spot. So the first character of the file can either be an "f," "g," or an"h." Let me explain what all of these symbols mean:

{f,g,h} The first character of the file can be one of these three characters. These characters correspond to the various fixes that can be applied to the loop data. The files with the "f" character mean that there has been no fix applied, the "g" means that the holes in the loop data were fixed, and the "h" means that the holes were fixed and the consistency errors were corrected. A complete discussion of the various fixes is given in Section 5.2. Which set of data gets generated is of course determined by the parameters in the runfile. In the rest of the discussion here I will assume that we are dealing with the "gloop" files because some of the calculations can only be done on the "gloop" files.

XX The "XX" mean the loop detector number. So the file name prefix "gloop7" corresponds to the loop data from loop 7 that has had the holes corrected.

{n,s} This corresponds to either the northbound or southbound detectors. For most loop stations there are southbound as well as northbound detectors. This character distinguishes between the two.

{s,c,o} This corresponds to the type of data that the file holds. The character "s" means speed, "c" means counts, or flows, and "o" means occupancies. So the file prefix "floop3.sc" means the flow data from the southbound direction of loop #3 that hasn't had any fixes done to it.

Y This last character can either be a number or the character "d." A number will correspond to either a specific lane or an on or off ramp. The character "d" means that the file is for the average over all lanes.

Let's take a look at a specific example. Let's say that we want the speed data from loop 4, northbound, lane 2. This means that XX, the loop number, will be 4, and Y, the lane number, will be 2. Finally since this is northbound speed data we have our final filename of:

```
floop4.ns2
```

I don't mean to mislead you to thinking that the individual lane files are easy to identify - they aren't. But the complications are discussed a little bit later on.

The **fsp** program also makes gnuplot executable files to facilitate viewing the various loop data values. The gnuplot executable filenames have a specific pattern to them so that you can recognize them. It is:

```
{f,g,h}loopXX.g{s,c,o}{v,p}
```

You'll notice that this looks a lot like the loop data files. It was done that way on purpose. The first thing that is different with the filename is that you now no longer specify whether you want the northbound or southbound. Instead you put a "g" in that spot. This is because each gnuplot executable makes plots of both the northbound and southbound values. The second thing that is different is that the last character no longer specifies the lane. Instead it specifies whether the gnuplot file will generate a plot to view or a plot to print. For example, if you wanted to view the occupancy data for the average over all of the lanes at loop 7, which is stored in the data files named `floop7.nod` and `floop7.sod` (assuming that you wanted the unfixed data), then you would need to use the gnuplot executable file named `floop7.gov`. To actually view the file you would type:

```
gnuplot floop7.gov
```

To print this file out to the printer you would just use the extension "gop" instead of "gov". Note that the gnuplot executable files can only generate plots of the average data, not of the individual lanes.

Examples of all of the loop plots are given in Section 15.5. Remember that the printer where the output goes is specified by the runfile variable `GNU_PRINTER`. So if you have

a favorite printer then you need to change this parameter in the runfile before you run the **fsp** program. If you have already generated the files and you want to choose a different printer without rerunning the program then simply edit the gnuplot executable file and substitute your new printer name for the old one.

Below is a listing of all of the loop plot file type extensions. The file prefix, when there is one, is enclosed in parentheses.

	PPS	SPD	OCC
Individual lanes (gloopXX)	.{n,s}cY	.{n,s}sY	.{n,s}oY
Summary (gloopXX)	.{n,s}cd	.{n,s}sd	.{n,s}od
GNU view 1 (gloopXX)	.gcv	.gsv	.gov
GNU print 1 (gloopXX)	.gcp	.gsp	.gop
GNU view all of 1 (gnuview)	.pps	.spd	.occ
GNU print all of 1 (gnuprint)	.pps	.spd	.occ
GNU view everything	gnuview.all		
GNU print everything	gnuprint.all		

To view a whole bunch of files right in a row then you would use the shell script gnuview.*, or gnuview.all. For example, if you wanted to view all of the PPS data right in a row without having to type in all of the individual commands you would type

```
gnuview.pps
```

Let's look at some examples of this naming scheme. Below is a listing of a certain portion of the output:

```
clair 1: ls floop6.*
floop6.gcp      floop6.gov      floop6.ncd      floop6.scd
floop6.gcv      floop6.gsp      floop6.nod      floop6.sod
floop6.gop      floop6.gsv      floop6.nsd      floop6.ssd
```

These are the output files that are generated from the data file `loop6.dat` when the following parameters are specified in the runfile:

```
LOOP_FLOW_PLOTS =       YES_CALC_ALL_FLOW_PLOTS
LOOP_TEXT =             LOOP_NO_REPORTS
```

Note that there are other parameters that you have to specify but these are the main ones. The question that I am trying to answer is, "What are all of these files?" Let's first look at just one file: `floop6.ncd`. As we said above the "f" in front of the word "loop" means that it's the unfixed loop data. The "6" means that it's for loop #6. And the extension, "ncd", means **N**orth **C**ounts **D**ata. The rest of the files follow the same naming convention:

```
floop6.ncd <- Northbound Counts Data
floop6.nod <- Northbound Occupancy Data
floop6.nsd <- Northbound Speed Data
floop6.scd <- Southbound Counts Data
floop6.sod <- Southbound Occupancy Data
floop6.ssd <- Southbound Speed Data
```

The gnuplot executable files follow a similar type of naming scheme:

```
floop6.gcp <- Gnuplot Counts Print (to printer)
floop6.gcv <- Gnuplot Counts View
floop6.gop <- Gnuplot Occupancy Print (to printer)
floop6.gov <- Gnuplot Occupancy View
floop6.gsp <- Gnuplot Speed Print (to printer)
floop6.gsv <- Gnuplot Speed View
```

So the gnuplot executable file `floop6.gcv` would display the plots of the counts data on the screen. It will first display the northbound plot and then it will prompt you for a return. When you have pressed return it then displays the southbound plot. If you had chosen the file `floop6.gcp`, meaning **G**nuplot **C**ounts **P**rint, then the two graphs would be printed out to the printer and the program would not prompt you for a return.

As was said above, if you wanted to view all of the PPS data files for the current directory then you can just use the file `gnuview.pps`. Note that the files that start with the key word "gnu", like gnuview and gnuprint, are not gnuplot executable files. They are shell script files that can be run from the command line by just typing their name.

Finally, as promised, the last thing to discuss about the loop files is the individual lanes files and the individual ramp files. When the program generates the files for the main line lanes it first goes through and pulls out all of the data for the individual lanes and places them in files. If the program is told to pull out the on ramp and off ramp data then it does this at the same time. Well, this generates a lot of files and as you can imagine, the naming problem becomes quite complex. Just to recap, for the individual lane files the basic filename is: VloopWW.XYZ. Where:

V: this is either "f," "g," or "h" depending on what gets fixed.

WW: this is the loop number (or cabinet number).

X: this is either "n" or "s" for the northbound data or the southbound data.

Y: this is one of:

c: means counts or pps.

s: means speed.

o: means occupancy.

Z: this is the lane number or the on or off ramp number. This is a little tricky so let me explain a little bit more.

The Z value is a value that starts at 1 with the first file name, which corresponds to the inside most lane, and then counts up with each successive filename, and each lane. The way that the files are ordered, or saved, is:

1. main line lanes
2. on ramps
3. off ramps
4. ramp demand detectors
5. ramp queue detectors

So the main line lane filenames get the lower numbers and the ramp queue detectors get the higher numbers. I don't want to lead you to believe that they get any arbitrary higher number - there is a set pattern. Let's look at an example.

If we had 5 main lanes, 1 on ramp, 2 off ramps, 2 demand detectors, and 2 queue detectors then the labeling scheme would go like this:

```
floopWW.XY1     - lane 1 data
floopWW.XY2     - lane 2 data
floopWW.XY3     - lane 3 data
floopWW.XY4     - lane 4 data
floopWW.XY5     - lane 5 data

floopWW.XY6     - on ramp 1 data

floopWW.XY7     - off ramp 1 data
floopWW.XY8     - off ramp 2 data

floopWW.XY9     - demand detector 1 data
floopWW.XY10    - demand detector 2 data

floopWW.XY11    - queue detector 1 data
floopWW.XY12    - queue detector 2 data
```

Notice how the "Z" parameter kept on increasing with each new file name. Let's look at a real example. If we wanted to take a look at cabinet #4 we would see that we have:

- Southbound: 4 main lanes, 1 off ramp

- Northbound: 4 main lanes, 2 on ramps, 2 demand, and 2 queue detectors

So the labeling scheme for the PPS data would be like this:

```
floop4.nc1     - northbound lane 1 data
floop4.nc2     - northbound lane 2 data
floop4.nc3     - northbound lane 3 data
floop4.nc4     - northbound lane 4 data

floop4.nc5     - northbound on ramp 1 data
floop4.nc6     - northbound on ramp 2 data

floop4.nc7     - northbound demand detector 1 data
floop4.nc8     - northbound demand detector 2 data

floop4.nc9     - northbound queue detector 1 data
floop4.nc10    - northbound queue detector 2 data

floop4.ncd     - northbound main line data aggregate

floop4.sc1     - southbound lane 1 data
floop4.sc2     - southbound lane 2 data
floop4.sc3     - southbound lane 3 data
floop4.sc4     - southbound lane 4 data

floop4.sc5     - southbound off ramp 1 data

floop4.scd     - southbound main line data aggregate
```

Notice how the "Z" parameter started back over from 1 when it switched to a different kind of file: it went back to 1 when it started generating the southbound files. Also, notice that if you don't know what the structure of the freeway is at a particular point then it is very hard to figure out from the filenames what is what. My point being that you have to know what the lane structure is before you can figure out what the filenames mean.

15.4 The Calculated Data Set

When I refer to the calculated data set I mean the data files that were calculated from the basic loop data. These files are used later on by the **fsp** program to calculate things like incident delay and to generate the contour plots. But since they are generated only from the loop data they are discussed here instead of in Chapter 16 where the rest of the cross data analysis output is discussed. There are two major types of calculated data, these are the delay files and the emission files. With each type data the **fsp** generates some raw data files and some LaTeX tables that can be processed with LaTeX and printed out on any postscript printer.

15.4.1 The Loop Delay And Density Files

The first set of calculated files are the delay and density files. There are a couple of different types of delay files. There are files that hold the data for each time period, or instantaneous files, there are files that hold the cumulative delay over the whole study section, and then there are files that hold tables of delay values.

15.4.1.1 Instantaneous Loop Files

The program calculates the delay, density and differential density for each time period for every loop detector. These values are then used later on to calculate the delay for each incident and to generate various contour plots. A short list of these files is given below:

1. The delay data files. These are the delay at each loop detector for each time period.
2. The density data files. These are the density of traffic at each loop detector for each time period.
3. The differential density data files. These are percentage that the current density is over the average density for that loop detector for every time period.

These files are referred to as data files because they hold only the raw calculated values and they have the standard **fsp** file format of two columns: the first column being the time in seconds since midnight and the second column being the value for that time. The file naming conventions for these three different files are exactly the same as for the basic loop data except for one character. So the naming scheme is:

```
{f,g,h}loopXX.{n,s}{t,d,e}d
```

Where the "t" stands for the delay files, the "d" stands for the density files, and the "e" stands for the differential density files. There are a couple of things that I should point out about these files:

1. The last character in the filename of these files is always "d" which means that the file is for the whole section of freeway, not for a single lane. These files are not generated for each individual lane.
2. These files were not meant to be viewed as individual files like the basic data set. Therefore, no gnuplot executable files are made to display them.
3. These files are used by a different section of the program to generate contour plots. The contour plots are described in Chapter 16.

15.4.1.2 The Cumulative Delay Files

Along with the instantaneous delay files the **fsp** program will also generate a cumulative delay file for the whole study section. The **fsp** program will calculate the total delay at each time period for the whole freeway by summing up the delay at each loop detector. It will then generate a file that has the cumulative delay over the whole time period. There is one file for each direction. The files are named according to the following format:

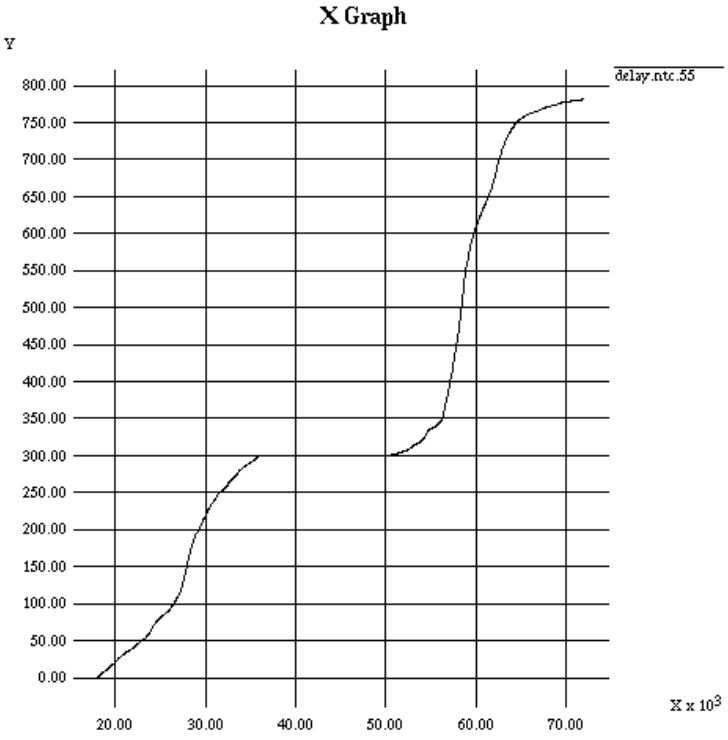

Figure 15.1: Cumulative Loop Delay.

```
delay.{n,s}tc.ZZ
```

Where the "ZZ" stands for the reference speed of the congestion. The files are stored with respect to the reference speed so that multiple runs of the **fsp** program can be run without overwriting the files. If the delay calculation is done with respect to the average speed then "ZZ" will be an "a."

The cumulative speed file can be viewed with **xgraph** in the following manner:

```
xgraph delay.ntc.55
```

This will generate a plot on your screen that looks like Figure 15.1. The horizontal axis of Figure 15.1 is the number of seconds since midnight (5am is 18000 and 2pm is 50400). The vertical axis is the cumulative delay in vehicle-hours. Notice that there is a spot in the middle of the graph from 36000 until 50400 that is flat. During this time there was no data collected and therefore there are no delay values. There is a line there because the **xgraph** program draws it's own line from point to point and it doesn't car if there is a gap in the data. If the time period of interest doesn't cover the dead zone then you won't have this affect.

15.4.1.3 The Loop Delay Tables

The **fsp** program also generates a LaTeX table of the delay values. Section 13.3 explains how to view and print any LaTeX table. Figure 15.2 is an example of one page of a table that was

Directory: lp021693

Time Slice: 5:01 to 5:02

Length of time slice: 1 minutes

Speed Threshold: 55 mph

Segment	Cabinet #	Length (mi)	# Lanes	Speed (mph)	Flow (vph)	# Vehicles	Delay
1	5	0.13	4	61.4	1312	22	0.000
2	15	0.64	5	63.0	1708	28	0.000
3	17	0.25	4	62.1	1220	20	0.000
4	4	0.13	4	62.5	1427	24	0.000
5	12	0.64	4	62.7	1366	23	0.000
6	13	0.31	4	64.2	1180	20	0.000
7	19	0.39	4	61.4	1159	19	0.000
8	18	0.27	5	61.2	1121	19	0.000
9	6	0.27	5	60.8	793	13	0.000
10	11	0.29	5	63.2	518	9	0.000
11	2	0.27	5	62.5	962	16	0.000
12	10	0.42	5	63.9	1190	20	0.000
13	20	0.29	5	65.1	869	14	0.000
14	7	0.32	5	63.8	1464	24	0.000
15	1	0.32	5	6.5	640	11	0.466
16	3	0.36	5	64.9	1525	25	0.000
17	16	0.37	5	60.2	1739	29	0.000
18	8	0.25	4	63.4	671	11	0.000
						TOTAL	= 0.466

Northbound Delay Calculation

Figure 15.2: Loop Delay Table.

generated by the **fsp** program. This table is for the time period from 5:01 am until 5:02 am in the northbound direction on February 16th, 1993. There is one table for each output period and each direction. The table in Figure 15.2 has at the top a few lines about the conditions under which the delay was calculated. We can see in this example that the delay was calculated with respect to a congestion speed of 55 mph. The table itself lists out the parameters and conditions at each loop detector and the calculated delay. You'll notice that the calculated delay for most of the loops is zero. This is because the congestion speed is 55 mph and most of the speeds on the freeway are higher than that. The number at the bottom that is labeled "TOTAL" is the sum of the delay over all of the loop detectors.

15.4.2 The Loop Emission Files

The second type of calculated data set are the emissions files. These files hold the emissions of carbon monoxide, hydrocarbons, and nitrogen compounds for the loop data. The nice thing about these files is that they are exactly like the loop files except for one character. The naming

scheme of these files is as follows:

```
{f,g,h}loopXX.{n,s}{g,h,n}Y
```

You can see that the file naming scheme is a lot like the regular loop files. The only difference is the second character in the file extension. The "g," "h," and "n," stand for the carbon monoxide, hydrocarbon, and nitrogen emissions, respectively. Some things to note about the emissions output:

- The emissions generation routine is exactly like the loop delay generation routine. You can calculate the emissions with respect to the average speed or with respect to a constant speed.
- The **fsp** program will generate emission LaTeX tables just like the loop data. These are named `emissions.{co,voc,no}.tex`. You need to process these the same way as the delay tables.
- A cumulative emissions file is also made (just like the cumulative delay files). The file naming scheme is `emissions.{co,voc,no}.{n,s}c.Z` where the "Z" is the reference speed at which the calculation was done.
- The emissions files are all placed in the individual loop day output directories.

15.4.3 The Aggregate Loop Files

One useful thing that the **fsp** program does[1] is it will calculate the total delay and the total number of vehicle-miles traveled for each individual loop directory. This is only done when the runfile parameter `LOOP_AGGREGATE_VALUES` is set to `YES_CALC_AGGREGATE_VALUES`. When this happens the following table is generated for each day:

```
Aggregate Delay:
   Loop data: /home/pal2/FSP/Set1/Loopdata/lp021693
   Southbound delay am, pm =    69.56,   851.71
   Northbound delay am, pm =   229.35,   423.51

Aggregate Vehicle Miles:
   Loop data: /home/pal2/FSP/Set1/Loopdata/lp021693
   Southbound veh-mi am, pm =   108856.43,   114284.34
   Northbound veh-mi am, pm =   113834.71,   113697.62
```

This simply gives the the delay and vehicle-miles traveled for each direction and each shift. Note that the delay is in vehicle-hours and that the vehicle-miles traveled is in vehicle-miles (who would have though?). There are some things that I should point out about the aggregate tables:

[1] Maybe the only useful thing.

- The aggregate values are only summed up over a specific time period. These time periods are from 6:30am until 9:30am, for the morning shift, and 3:30pm until 6:30pm, for the evening shift. These were the times that we had probe vehicles driving around on the freeway.
- The routine that does the aggregate calculation assumes that the proper loop files have already been calculated. If they haven't then all of the results will be zero.
- The aggregation routine can only be run on the loop files that have had the holes fixed. The **fsp** program will not allow it to run on the unfixed loop data.
- The aggregate tables are only printed out to the screen - there are never saved to a file. If you want to save them then you'll need to either cut and paste with the mouse or redirect the output of the **fsp** program to a file.

After processing all of the loop data the aggregation routine will generate a table that sums up all of the values. A sample of one of these tables is given below:

```
Total Aggregate Delay:
   Southbound total delay am, pm =     302.35,    1083.40
   Northbound total delay am, pm =     562.49,     917.46
   Overall total delay           =    2865.70

Total Aggregate Flow:
   Southbound total veh-mi am, pm =   208387.78,   224509.69
   Northbound total veh-mi am, pm =   221243.72,   224889.83
   Overall total flow             =   879031.02

Total Delay per 10^6 vehicle miles:
   Southbound total delay/10^6 veh-mi am, pm =    1450.91,    4825.62
   Northbound total delay/10^6 veh-mi am, pm =    2542.39,    4079.58
   Overall total delay/10^6 veh-mi           =    3260.06
```

The total aggregate values are simply the summation of the various values for all of the days. The only thing that is new here is the delay per million vehicle-miles. We thought that this was an interesting statistic so we had the program calculate it. You can relate this to the delay caused by so many incidents per million vehicle-miles.

15.5 The Loop Plots

Table 15.2 is a print out of the six different plots from one loop. They could be generated by using the following commands:

```
gnuplot floop6.gcv
gnuplot floop6.gsv
gnuplot floop6.gov
```

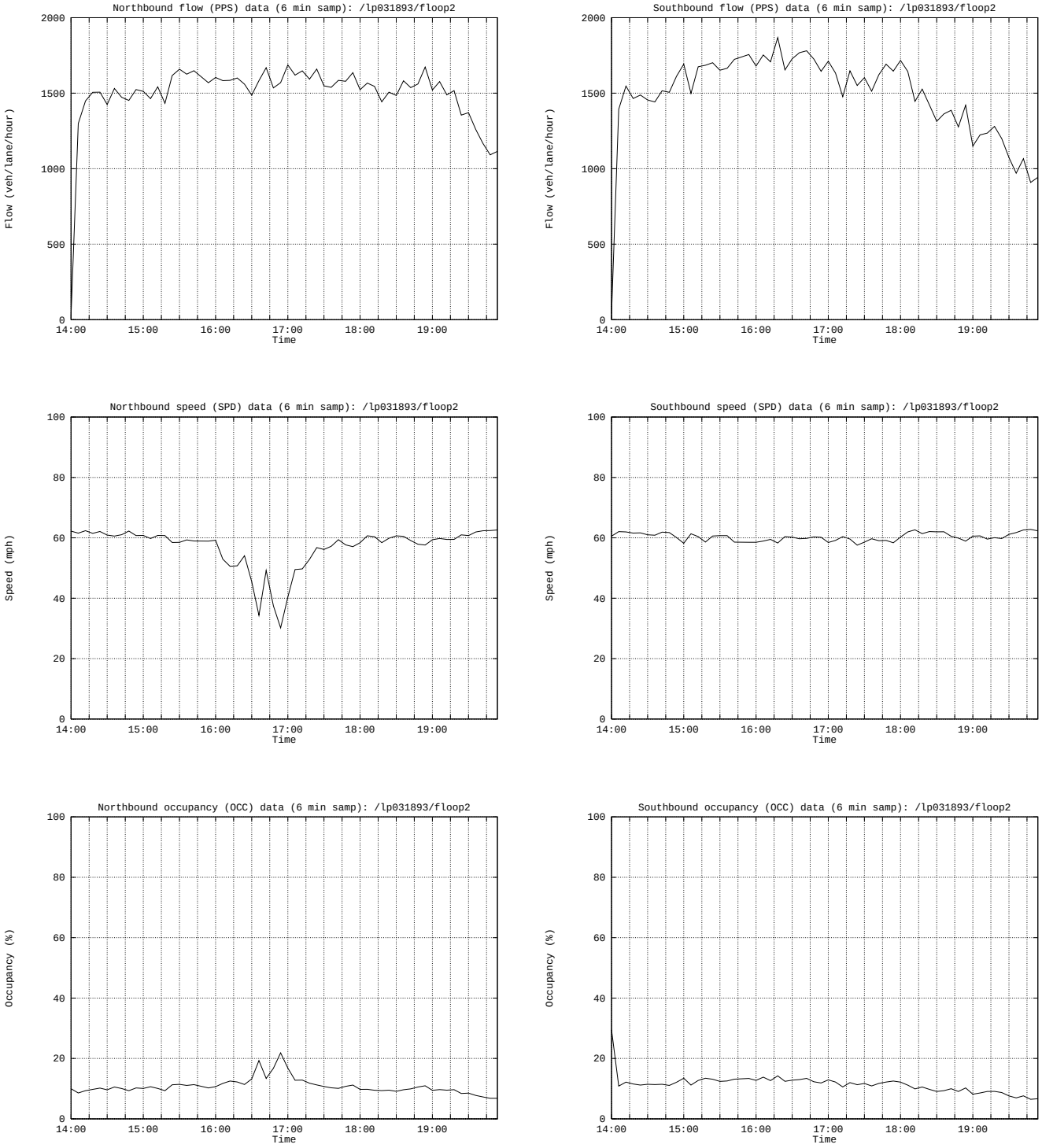

Table 15.2: Loop plots.

Chapter 16

Program Output: The Incident and Cross Data Analysis

Probably some of the most interesting output that the **fsp** program generates comes from the incident data. Since one of the main goals of the program was to be able to calculate the delay per incident there is a lot of emphasis on the incident output. As a matter of fact, the default setup for the program is to only print out incident statistics and to hide all of the loop and car data output. The **fsp** program generates quite a bit of output that deals with the incidents. It reports on the correlation between the incident database and probe vehicle trajectory data. It reports on the various fixes that are applied to the incident data to make it more accurate. It reports on the delay calculations that are performed for each incident. It reports on quite a few things. But the point that I am trying to make is that most of the reports made on the incident involve one of the other data sources. As a result, it is hard to talk about the output that only deals with the incidents without mixing in something else. As a result, this chapter will discuss the incident output and the cross data analysis output.

16.1 Quick Overview Of The Incident And Analysis Output

When the **fsp** program processes the incidents it basically has two things that it can report on. These are the various fixes that it has applied to the incident data, and the incident delay calculations. As was shown before, Figure 16.1 show us a simple conceptual picture of what the program does to fix the incident data. Depending on what the user tells the program to do, it can attempt to fix the incident data directory from the probe vehicle data or it can fix the incident data from the already generated incident fix files. Whichever option the user chooses the program will want to report on how it did attempting to fix the incidents.

For the delay calculations, the program would like to report not only on what the delays were for each incident, but also how it got those delay. So there is some text that is printed to the screen reminding them of the delay calculation conditions. The program will also generate some graphs of the analysis. A review of what this process looks like can be seen in Figure 16.2. In that figure you can see that there are a few different graphs that the program will generate: histograms, cumulative distributions, and incident delay versus incident duration. Finally, the contour plots are also generated in the incident processing section. The reason that

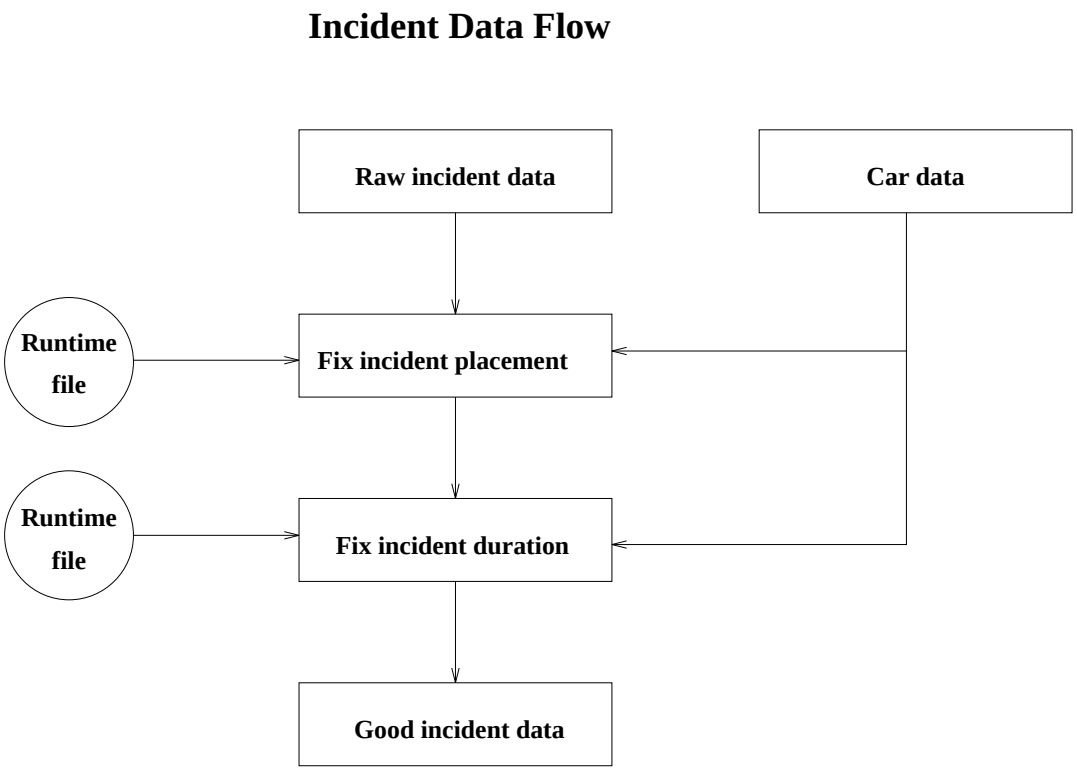

Figure 16.1: Data Flow For Fixing The Incidents.

this is done here instead of in the loop data processing section is because the contour plots have the incidents on them. In order for the program to know what the incidents are it first needs to filter them out and process them.

16.2 Textual Output

The **fsp** program gives the user many options on how much text they would like to see with the incident and data analysis output. These options are set by the runfile parameters `INC_RAW_OUTPUT_LEVEL` and `INC_FINISHED_OUT_LEVEL`. The various categories of output are listed out below:

- ○ Working directories.
- • Incident filter listing.
- • Initial number of matched incidents.
- • Basic characteristics of initial incidents.
- • Complete characteristics of initial incidents.
- ▷ Incident database - probe vehicle correlation results.
- ⊙ Incident duration fix results.
- ⋆ Incident distribution over the various days.
- ⋆ Delay calculation conditions.

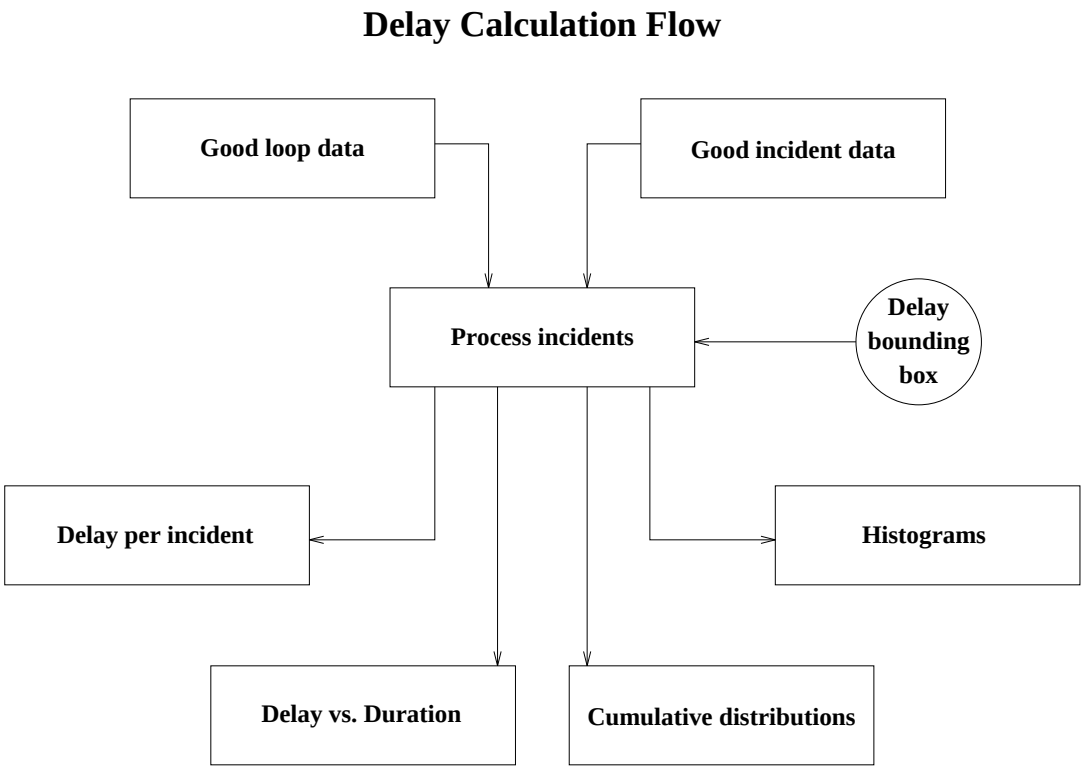

Figure 16.2: Generating The Incident Delays.

- ⋆ Vital characteristics of finished incidents.
- ⋆ Incident space-time boxes for delay calculation.
- ⋆ Incident queue length analysis.
- ⋆ Delay calculation results.

The item that is listed with a "∘" is something that always shows up no matter how you set the options. This can be considered the base case output. The items that are listed with a "•" are generated depending on how the parameters are set for the raw incident output. The item with a "▷" beside it is generated by the correlation analysis. If you run the correlation analysis then this will be generated. The incident duration fix results - the item with the ⊙ beside it - are generated when you attempt to fix the incident durations either from the probe data or from a premade file. Finally, the items listed out with a "⋆" are items that are generated depending on how the parameters are set for the finished incident output. In the subsections that follow we will discuss first the base case output, then the raw incident output, then the incident database - probe vehicle correlation results, and finally the finished incident output options.

16.2.1 The Base Case Output

If you are processing the incident data then you aren't guaranteed to get any output at all. The runfile parameters `INC_RAW_MATCH_OUTPUT` and `INC_FINISHED_OUTPUT`, which I will call

the incident output control parameters, dictate if you will get any output. If these are both set to indicate that no output should be generated then the only thing that will come up on the screen is the following:

```
clair 1: fsp Runfiles/rf11111.run Incidents/my_inc.dat 0
Start of fsp
Using the following data directories:
    Loopdata directory = /home/pal2/FSP/Set1/Loopdata
    Cardata directory = /home/pal2/FSP/Set1/Cardata
    Incident directory = /home/pal2/FSP/Set1/Incidents

    Data output directory = /home/pal2/FSP/Out5min
    Loop output directory = /home/pal2/FSP/Out5min/Loopdata
    Car output directory = /home/pal2/FSP/Out5min/Cardata
    Incident output directory = /home/pal2/FSP/Out5min/Incidents


******              END OF PROGRAM              ******
```

This is not very interesting. This simply reminds the user what directories the **fsp** program is using to read in data and where it will place the output. For analyzing the incident data this is worthless. So it is imperative that you set the two incident output control parameters such that you actually get some output.

16.2.2 The Raw Incident Output

The raw incident output is generated from the incidents that justed passed through the incident filter. No fixes have been applied to them and their delays have not been calculated. This is sort of a first check to make sure that everything is going ok. In order to get any output here at all the raw incident control parameter, `INC_RAW_MATCH_OUTPUT`, needs to be set to something other than `NO_RAW_MATCH_OUTPUT`. We will assume that it is set to `SCREEN_RAW_MATCH_OUTPUT` so that the output comes up on the screen. If this was set to `FILE_RAW_MATCH_OUTPUT` then the results would have been placed in the file named `inc.matched.raw` in the incident output directory. For a complete listing of the various options see Chapter 7.

If the raw incident output level parameter, `INC_RAW_MATCH_OUTPUT`, is set to `INC_RAW_OUT_SPARSE` then the raw incident output section will look something like this:

```
Matching incident structure:
Field   Heading              Values
   0    DATA_TYPE              F
   2    DATE                   2/16/93 -  2/17/93

Incidents matched: 46
```

This will list out all of the fields that the program is using to filter incidents with. In this case the incident filter is looking for field data that occurred only on 2/16/93. Remember that in

the incident filter in order to specify one date you have to have the start date and the end date be one day apart (like they are above). It will also list out the number of incidents that made it through the filter.

If INC_RAW_MATCH_OUTPUT is set to INC_RAW_OUT_MEDIUM then in addition to the listing above there will be a listing that contains a few basic characteristics of each incident. It will look something like this:

```
Stats on raw incident match:
Incident #    Date     Time   South   Link #
      1     2/16/93    6:49     1       3
      2     2/16/93    6:54     1       3
      3     2/16/93    7:00     1       8
```

You can see that this listing is only very basic. It only lists out the date, time, direction, and link number of the incident. Note that the column labeled "South" is simply the field in the incident database that holds the direction. So a 1 in this column means that it's in the southbound direction and a 0 means that it's in the northbound direction.

If INC_RAW_MATCH_OUTPUT is set to INC_RAW_OUT_VERBOSE then instead of the sparse listing above, the program will print out all of the fields of every incident. The start of this output looks something like this:

```
Field   Heading              Value Meaning
   0    DATA_TYPE              F   Field data
   1    INC_NUMBER             1   Incident number
   2    DATA_TYPE        2/16/93   Date
   3    SHIFT                  0   AM Shift
   4    TIME                6:49   Time
          .
          .
          .
```

If you are filtering out more than a just a few incidents then this is probably so much output that it's not useful. Actually, there probably isn't much need to ever use anything more than the sparse output on the raw data - you can get everything else that you need from looking at the finished incident output.

16.2.3 Incident Database - Probe Vehicle Correlation Results

The incident database - probe vehicle correlation is done when the program is trying to figure out the correct locations for the incidents. The program reads in the probe vehicle data and records the locations of all of the key presses. It then reads in the incident database and attempts to match up the incidents with the key presses. When the program is finished with that process it generates some text to inform the user of the results and it generates some graphs so that the user can see how the correlation turned out. The graphs are discussed in Section 16.3.2 below.

The correlation routine will generate two levels of text depending on the setting of the finished incident output level parameter INC_FINISHED_OUT_LEVEL. If this runfile parameter

is set to INC_FIN_OUT_SPARSE then the output from the correlation routine will look like the following:

```
Correlation Results:
Incident database match statistics:
   Total: # incidents, # covered, ratio = 46, 46, 100.0%
   Total: # time entries, # matched, ratio = 151, 89, 58.9%
   Total: # covered incidents, # changed, avg change(ft) = 46, 29,  1160
   Total: # with new loop, ratio = 8, 17.4%

Probe vehicle match statistics:
   Total number of entries = 154
   Number of entries in study section = 117
   # matched entries in study section = 89
   ratio = 76.1%
```

The first thing that you will note is that there are two types of statistics: the incident database match statistics and the probe vehicle match statistics. When the correlation routine tries to match the two data sets up it keeps statistics on how well each data set matched the other. These values might seem a little confusing at first so a line by line explanation is given below. The first set of statistics that I will explain deal with the incident database:

```
   Total: # incidents, # covered, ratio = 46, 46, 100.0%
```

This is the number of incidents that made it through the incident filter. The number of covered incidents is the number of incidents that had some probe data that coincided with it that was specified in the runfile. In this case, every incident had some probe data that covered the same time period. If, for example, in the incident filter you specified that you wanted to look for the incidents that occurred on February 19th, but in the runfile you only specified the car data for March 3rd then these two data sets would not overlap. In that case, none of the incidents would have been covered.

```
   Total: # time entries, # matched, ratio = 151, 89, 58.9%
```

For each incident in the incident database there is a listing of the number of times that the incident was witnessed by the probe vehicle drivers. The number of time entries here, is the summation, over all of the incidents, of the number of times each incident was witnessed. The number matched is the number of these entries that the computer matched with a specific key press from the probe vehicles, and the ratio is just the percentage that were matched.

```
   Total: # covered incidents, # changed, avg change(ft) = 46, 29,  1160
```

The number of covered incidents here is the same as in the first line - it's just the number of incidents that had coinciding car data with them. The number changed is the number of incidents that had their location changed due to the correlation routine. The "avg change" is the average amount of change in feet for all of the incidents.

```
Total: # with new loop, ratio = 8, 17.4%
```

This is the number of incidents that were changed enough to have a new loop number assigned to them. If this number is zero then the correlation with the car data had no effect. If this number is zero then it could be that the key presses all matched up exactly with where the drivers said that incidents occurred. Unfortunately, a more likely explanation is that all of the key presses were outside of the incident boxes.

This second set of statistics deal with the probe vehicle data:

```
Total number of entries = 154
```

This is the total number of times that the drivers in the probe vehicles hit a key indicating that they were driving past an incident.

```
Number of entries in study section = 117
```

This is the number of entries that fell within the study area on the freeway. There was a section of the road that already had the Freeway Service Patrol tow trucks operating on it and so we took all of the incidents in this area out of the database. Therefore, we should exclude the key presses that show up in that area as well. This number should be the same as "# time entries" in the incident statistics above but it usually isn't.

```
# matched entries in study section = 89
```

This is the number of key presses that the computer thinks it can match up with driver recorded witness points in the incident database. This number is the same as "# matched" in the incident statistics above.

```
ratio = 76.1%
```

This is the ratio of the number matched to the number of entries in the study section (this is still only within the probe vehicle data). We usually use this as a measure of how good our fit is. The closer this number is to 100% the better.

If the finished output level parameter is set to `INC_FIN_OUT_MEDIUM` or `INC_FIN_OUT_VERBOSE` then the output from the correlation routine, in addition to the output described above, will include the following table:

```
SUMMARY of all COVERED incidents:
   Inc #  # Entries # Matched  Dist(ft) Loop  New Dist(ft) New Loop  Change
    205       26        19      35640    12      35764        13       0.02
    210        3         2      21040    20      20474         7      -0.11
    213        3         3      41360    17      38824        17      -0.48
    216        2         0      39600     4      39600         4       0.00
    219        2         3      47770     5      47619         5      -0.03
```

This table lists out for each incident the number of entries in the incident database, the number of those entries that got matched up with a key press in the probe vehicle data, the original location (in feet) of the incident[1], the original loop detector of the incident, the new incident location and loop detector, and finally, the change in miles between the old distance and the new. Note that although the distance may change this doesn't necessarily mean that there will be a new loop detector. Also note that the distance in feet is from the starting point of the probe vehicle run. Since the probe vehicle starts in the southbound direction first, if an incident occurs on the northbound section then the distance to that incident is the distance from the starting point all the way to the turn-around point and then back up (north) to the incident. The final thing to notice is that if there are more key presses in an incident box than there are entries in the incident database then the program will still allow those key presses to be matched to that incident. You can see that this happened for incident #219 above.

Finally, if the finished output level parameter is set to `INC_FIN_OUT_VERBOSE` then the correlation routine will print out one additional table. This table is an incident location shift table than can be used as the new incident location fix file. This table is only generated when the runfile parameter `FIX_INC_LOCATION` is set to `YES_FIX_INC_DELAY`. A small sample of one of these tables is given below:

```
Possible new incident location fix table:
  Incident    Shift (miles)
     205      -0.48
     210      -0.36
     213      -0.48
     216       0.00
       .
       .
```

A complete example of how to use this table is given in Section 12.9. The basic idea is that by using these values in a new incident location fix file you could possibly reduce the effect of the correlation routine nothing. This is very handy if you don't want to have to process the car data and run the correlation routine each time that you want to get accurate incident locations.

One thing that you will notice about the correlation statistics is that they should probably have been generated for each shift and not for the study period as a whole. While that may be true, we still needed an overall measure of how well the correlation routine was doing. As a result we opted for the large measure instead of a bunch of smaller measures.

16.2.4 The Incident Duration Fix Output

The incident duration fix output is only generated when the **fsp** program is attempting to fix the incident durations by looking at when the different probe vehicles drove by an incident and didn't witness it. It doesn't matter if the duration fix is being derived from the probe data or from a runtime file, this table will alway be generated. The methodology behind this type of duration fix is discussed in Section 5.3.2 and an example of how to do this with a runfile is

[1]This location is the location after the incident location fix has been applied

given in Section 12.10. A sample of the table that the incident duration fix routine generates is given below:

```
Incident duration correction statistics:
              Before    New            After    New              Final
Inc    Start   Start   Start    End     End     End   Duration  Duration
 205   23880   23794   23837  34140   34150   34145    10260     10308
 210   27180   26717   26948  28260   28829   28544     1080      1596
 213   30060   29686   29873  30780   31152   30966      720      1093
```

Besides the incident number, which is in the first column, all of the values in this table are in seconds. The second column, which is labeled "Start," is the starting time of the incident that is recorded in the incident database. The column right after it, which is labeled "Before Start" is the last time before an incident occurred when any probe vehicle drove by the incident location and didn't witness the incident. The column labeled "New Start" is the adjusted start time that should be between the original start time and the time when the previous probe vehicle passed the incident location. The next three columns, columns 4-6, deal with the incident ending time. They follow the same format as the start time columns: "End" is the incident ending time, "After End" is the first time a probe vehicle drove by and didn't witness the incident, and "New End" is sometime between the two. The column labeled "Duration" is the original duration and the "Final Duration" is the new duration.

This table is only generated when the finished incident level parameter, `INC_FINISHED_OUT_LEVEL`, is set to `INC_FIN_OUT_MEDIUM` or `INC_FIN_OUT_VERBOSE`. This table is placed either on the screen on in the finished incident output file as determined by the parameter `INC_FINISHED_OUTPUT`.

16.2.5 The Finished Incident Output

The finished incident output is generated from the incidents after all of the fixes have been applied and the delay for each incident has been calculated. To get output for the finished incidents the finished incident control parameter, `INC_FINISHED_OUTPUT` needs to be set to something other than `NO_FINISHED_OUTPUT`. We will assume that it is set to `SCREEN_FINISHED_OUTPUT` so that the output comes up on the screen. If this was set to `FILE_FINISHED_OUTPUT` then the results would have been placed in the file named `inc.finished` in the incident output directory. For a complete listing of the various options see Chapter 7.

If the finished incident output level parameter, `INC_RAW_MATCH_OUTPUT`, is set to `INC_FIN_OUT_SPARSE` then the finished incident output section will contain three distinct parts: the incident distribution over the various days, the delay calculation conditions, and the delay calculation results. The incident distribution section is simply a listing of the number of incidents that occurred on each day during the study period and whether or not they occurred during the morning or the afternoon shift. A section of this table is given below:

```
Incident count by day:
     Date       #Incs/Day    #Incs/Shift
    2/16/93         46       AM    24     PM    22
    2/17/93          0       AM     0     PM     0
       .
```

```
    3/19/93          0       AM     0     PM     0
Average # incs:    1.9       AM   1.0     PM   0.9
```

Note that all of the days of the study section are listed out even if there aren't any incidents on those days (I cut out a large section in the middle for space considerations). Since in this example we are only looking at incidents on 2/16/93, nothing shows up on any of the other days.

The next piece of text that is generated under the finished incident routine is a listing of the various conditions that were used to calculate the delay for the incidents. This section looks like this:

```
Delay Calculation Conditions:
   Incident Explanation     = All incidents
   Car Headway              = 420 seconds
   Delay Calculation        = WRT_AVERAGE_SPEED
   Delay Calc Type          = ONLY_HAVE_POSITIVE_DELAY
   Congestion Speed         = 55 mph
   Fix Loop Holes Option    = YES_FIX_HOLE_ERRORS
   Incident Delay Method    = Fixed number of detector:
       Num Upstream Det.    = -1
       Num Downstream Det.  = 0
```

The incident explanation is the user supplied title that is put on the incident graphs that are discussed in Section 16.3.1. The car headway is the average headway time between the probe vehicles. Whether this gets used in the calculation is determined by the line below labeled "Incident Delay Method." If this line says "Fixed number of detectors," like it does now, then the car headway, along with a fixed number of detectors, is used to define the the incident space-time box that the delay is calculated from. If the "Incident Delay Method" instead says "Predefined space-time boxes" then the user defined space-time boxes are used and the headway time is ignored. The rest of the lines are pretty straight forward.

The last piece of output generated by the finished incident section under the output level parameter of `INC_FIN_OUT_SPARSE` is the delay calculation results. This is probably the section that will be used the most. It contains summary information on quite a few things:

```
Stats on all incident delays:
   Match incidents witnessed once = YES
   ALL results include headway time of = 7:00
   Number of Incidents   = 46
   Number witnessed once = 23
                         Min     Max     Mean     Std. Dev.    Std. Error
   Incident Duration       7     165     27.80      43.72         6.45
   TT Response  (  4)       0      37     16.25      18.50         9.25
   TT Clearance (  3)       0      28     17.33      15.37         8.88
   Incident Delay        0.00  446.74     22.77      69.84        10.30
```

```
Incident delay vs. duration line:
   Slope           =      0.915
   Intercept       =     -2.671
   Sigma_slope     =      0.197
   Sigma_intercept =     10.149
   Chi^2           = 147485.766
```

The first two lines tell the user whether the program attempted to match the incidents that were only witnessed once and what headway time the program used. The third line is the total number of incidents and the fourth line is the number of incidents that were witnessed once. The table below that gives the minimum, maximum, mean, standard deviation, and standard error for incident duration, tow truck response, tow truck clearance, and incident delay. Note that the numbers after the tow truck response and clearance headings are the number of each type. So in the table above, there were 4 incidents that had a tow truck respond, and the incident database had the tow truck clearance time for 3 incidents. The statistic that we used the most was the mean incident delay. The incident delay versus duration section gives the results of an attempt to fit a straight line to a plot of incident delay versus incident duration. This plot is discussed in Section 16.3.1.

These are the basic incident and data analysis statistics. The two other levels of the finished incident control parameter simply give a more detailed picture of the fixes that were done on the incident data and the delay calculation.

If the finished incident output level parameter, `INC_RAW_MATCH_OUTPUT`, is set to `INC_FIN_OUT_MEDIUM` then in addition to the output above, the finished incident output section will contain a table that lists out all of the vital statistics on the incidents. A section of this table looks like this:

```
Individual incident statistics:
              Inc. Type                          Good   Bad
Inc #   Date   D 1 2 3   Time South Link Loop  Files Files  Duration     Delay
   1  2/16/93  0 5 0 0   6:49   1    5    20     5     0     0:00:00      1.54
   2  2/16/93  0 3 0 0   6:54   1    4     7     4     0     0:27:00      4.36
   3  2/16/93  0 5 0 0   7:00   1   17     5    17     0     0:06:00      1.32
   4  2/16/93  0 0 2 0   7:34   1   13    12    13     0     0:00:00      0.58
```

The various columns to this table are as follows:

Inc # : This is simply the incident number.

Date : This is the date that the incident took place.

Inc. Type D : This is field **P** in the incident database that is labeled "Incident Type" in Section 4.5 and has the incident filter parameter `INCIDENT_DESCRIPTION`. This field basically tells you whether or not the incident is debris or not.

Inc. Type 1 : This is field **Q** in the incident database that is labeled "Type 1" in Section 4.5 and has the incident filter parameter `INCIDENT_TYPE_1`. This field tells you whether or not the incident is a breakdown or not.

Inc. Type 2 : This is field **R** in the incident database that is labeled "Type 2" in Section 4.5 and has the incident filter parameter `INCIDENT_TYPE_2`. This field tells you whether or not the incident is an accident or not.

Inc. Type 3 : This is field **S** in the incident database that is labeled "Type 3" in Section 4.5 and has the incident filter parameter `INCIDENT_TYPE_3`. This field tells you whether or not the incident is a CHP ticketing incident.

Time : This is the time that the incident was first witnessed by a probe vehicle.

South : This is a 1 if the incident occurred in the southbound direction and a 0 if the incident occurred in the northbound direction.

Link : This is field **I** in the incident database that is labeled "Link Identity" in Section 4.5 and has the incident filter parameter `LINK_ID`. This simply tells you the rough location of the incidents.

Loop : This gives you the loop number that is just upstream of the incident.

Good Files : If you are attempting to calculate the incident delay based on the non-fixed loop data then certain loop delay files might not be present. When the program reads in the loop delay files to calculate the incident delay it realizes that some of the files might not be there. This column gives the number of files that it did find.

Bad Files : This is the counterpart to "Good Files." This gives the number of files that it attempted to find and couldn't. If you are allows working with the corrected loop data then this column should always be zero. If it isn't then something is wrong.

Duration : This is the uncorrected duration of each incident.

Delay : This, of course, is the incident delay in vehicle- hours.

The table above is probably all that you'll ever need when working with the incident delays. But, if for some reason you would like to have an even more detailed analysis of what the program has done then you can set the finished incident output level to `INC_FIN_OUT_VERBOSE`. This will give you two more pieces of information. The first is another table that has some extra fields in it:

```
Summary of fixes to incidents:
                                    Fixed    Fixed    Fixed    Start End
Inc #   Start     End       Dur.    Start     End      Dur.  Lp  Lp  Lp  Delay
   1   6:49:00  6:49:00  0:00:00  6:45:30  6:52:30  0:07:00 20  16  20    1.54
   2   6:54:00  7:21:00  0:27:00  6:50:30  7:24:30  0:34:00  7  16   7    4.36
   3   7:00:00  7:06:00  0:06:00  6:56:30  7:09:30  0:13:00  5  16   5    1.32
   4   7:34:00  7:34:00  0:00:00  7:30:30  7:37:30  0:07:00 12  16  12    0.58
```

This table basically lists out the final space time box that the program uses to calculate the delay for each incident. An explanation of each column is given below:

Inc # : This is simply the incident number.

Start : This is the original starting time of the incident.

End : This is the original ending time of the incident.

Dur. : This is the original duration of the incident.

Fixed Start : This is the fixed starting time of the incident.

Fixed End : This is the fixed ending time of the incident.

Fixed Dur. : This is the fixed duration of the incident.

Lp : This is the number of the loop detector that is just upstream from the incident.

Start Lp : This is the number of the loop detector upstream of the incident that the program will start to calculate the delay from.

End Lp : This is the number of the loop detector downstream of the incident where the program will stop calculating the delay.

Delay : This is the incident delay in vehicle-hours.

The second piece of information that the program will generate under the verbose setting is a short analysis of the queue length for each incident. The program will generate a table for each incident that has the total cumulative delay versus the number of detectors upstream of the incident. A typical table would look like this:

```
Analysis of incident delay/queue length:
Detector Back   Delay

Incident #: 36
      1         4.90
      2         6.48
      3         6.92
      4        16.11
      5        20.88
      6        21.65
      7        21.66
      8        21.66
      .
      .
```

Note that the column "detector back" is just the number of detectors upstream of the incident - it is not the incident detector number. For this incident you can see that the delay tapered off to zero right after the seventh detector upstream from this incident. This probably means that the queue only extended back to the seventh detector as well. I must strongly warn you to look at the contour delay plots when attempting to interpret these tables.

16.3 Graphical Output

The **fsp** program will generate a couple of different types of incident and data analysis graphs. These are:

- Incident histograms.
- The incident delay versus duration.
- The correlation plots.
- The contour plots.

The first two types of plots, the incident histograms and the delay versus duration plot, are placed in the incident output directory. The correlation plots are placed in the individual car shift output directories and the contour plots are placed in the loop data output directories.

16.3.1 The Incident Plots

If the user sets the runfile parameter `INC_FINISHED_GRAPHS` to `YES_FINISHED_GRAPH` then the **fsp** program will generate four different plots:

- A histogram plot with the number of incidents.
- A histogram plot with the percentage of incidents.
- A cumulative distribution plot.
- An incident delay vs. duration plot.

There are a couple of runfile parameters that determine how these plots turn out. These are:

INC_EXPLANATION : This is an explanation that is placed at the top of all of the plots. There is some other information that is also placed in the plot title that the user has no control over. The maximum length that you should probably make any title is around 50 characters.

INC_GRAPH_MAX_NUM : This sets the range of the y-axis on the histogram plot with the number of incidents. This allows the user to set the range to a pleasing level. The reason that we don't let gnuplot set this is because gnuplot will always set the range to the maximum value. If we want to compare two different plots side-by-side then this can be very irritating. Unfortunately, what this means is that you have to first run the **fsp** program to see what the appropriate ranges should be and then come back and set the ranges in the runfile and then run the program again.

INC_GRAPH_MAX_PERCENT : This sets the range of the y-axis on the histogram plot with the percentage of incidents. This is just like the parameter above.

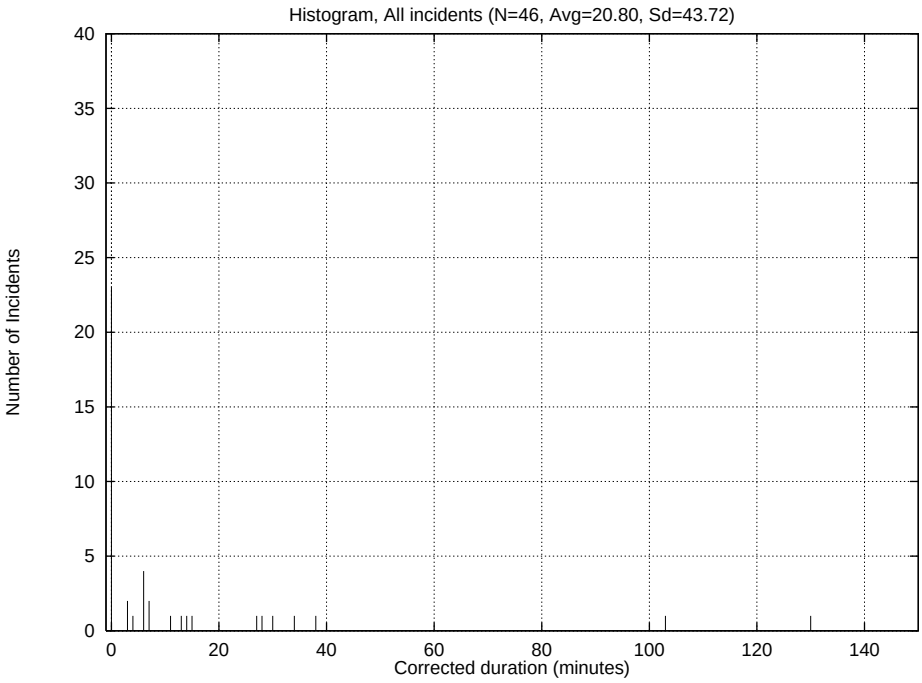

Figure 16.3: Histogram Of The Number Of Incidents.

A sample of each one of these plots is given on the next couple of pages by Figures 16.3 - 16.6. There are a couple of things to note about generating these plots:

- The maximum duration that an incident can have is set in the program to be 150 minutes - just over two hours. Any incident that falls outside of that range is discarded. Since this length of time covered our study period this shouldn't be too much of a problem. The disconcerting thing is that the **fsp** program spits out the following error message each time that it has to discard an incident:

  ```
  ERROR: Incident histogram overflow: Have to discard an incident.
  ```

  You should just ignore these messages.

- The line that is draw on the delay vs. duration plot is only a linear fit of the data. There no is evidence to suggest that the relationship between the duration and the delay should be linear.

The filenames that are used to save these plots in the incident output directory are as follows:

chist.dat.X : This is the data file for the cumulative histogram plot.

del.dur.dat.X : This is the data file for the delay versus duration plot.

fhist.dat.X : This is the data file for the histogram of the percentage of incidents.

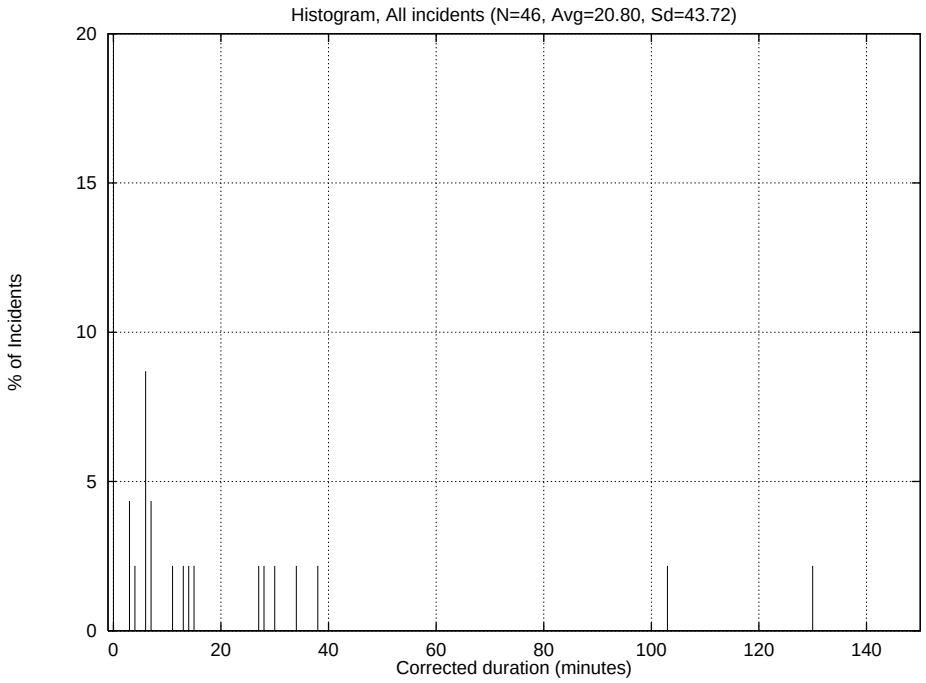

Figure 16.4: Histogram Of The Percentage Of Incidents.

gnuprint.ddX : This is the gnuplot executable file that will print the delay versus duration plot out to the printer.

gnuprint.hX : This is the gnuplot executable file that will print the histogram plots out to the printer.

gnuview.ddX : This is the gnuplot executable file that will allow the user to view the delay versus duration plot.

gnuview.hX : This is the gnuplot executable file that will allow the user to view the various histogram plot.

hist.dat.X : This is the data file for the histogram of the number of incidents.

The "X" in each of the file names above represents the incident file number or run number. Each time that the **fsp** program is run a run number is passed to it. This run number is used to index all of the output files so that the user can save multiple runs without having to copy files around. If you are using the **xfsp** program then this is taken care of automatically.

16.3.2 The Correlation Plots

The correlation plots are generated as a results of the program attempting to correlate the incident database and the probe vehicle data. A complete discussion of how the correlation plots are generated is given in Section 5.3.1. A typical correlation plot is given in Figure 16.7. The useful thing about the correlation plots is that they can be used to adjust the incident

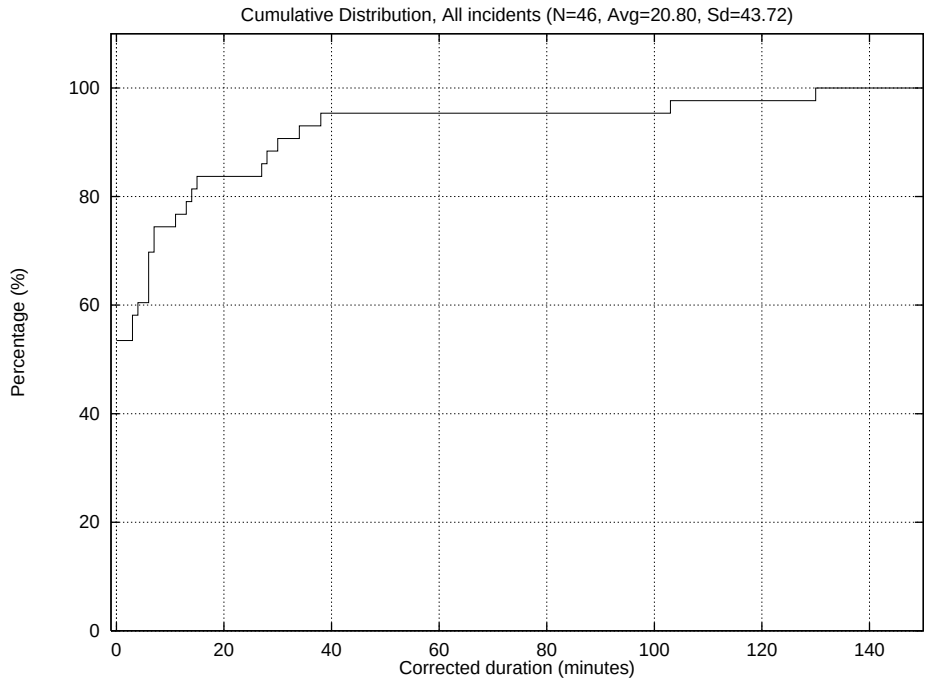

Figure 16.5: Cumulative Distribution Plot.

location via the incident location fix runtime file. An example of this is given in Section 12.9. There are a couple of runfile parameters that need to be set in order for the correlation plots to be made:

- Since the correlation routine is part of the incident processing section you need to tell the **fsp** program to process the incidents. This is done by setting the runfile parameter `PROCESS_INCIDENTS` to `YES_PROC_INCIDENTS` .
- The correlation routine is only turned on when the parameter `CORRELATE_CARS_DATABASE` is set to `YES_CORRELATE`. The correlation routine needs to be turned on in order for any graphs to be made.
- The runfile parameter `INC_CORRELATION_GRAPH` needs to be set to `YES_INC_CORR_GRAPHS`. This will tell the correlation routine to generate the graphs.
- Whether the incident numbers are placed on the plots is determined by the runfile parameter `NUMBER_INC_CORR_GRAPHS` . Even though you probably want to have the incident numbers on the plots, if there are too many incidents then it can get a little hard to see which number goes with which box. If this parameter is set to `YES_NUMBER_INC_CORR_GRAPHS` then the numbers will be placed on the plots.
- You have to process some car data so that the correlation routine will know where the key presses are. This means that you have to specify some car data and you have to tell the program to look for the incident key presses in the car data by setting the runfile parameter `INCIDENT_POINTS` to `YES_INCIDENT_POINTS`.

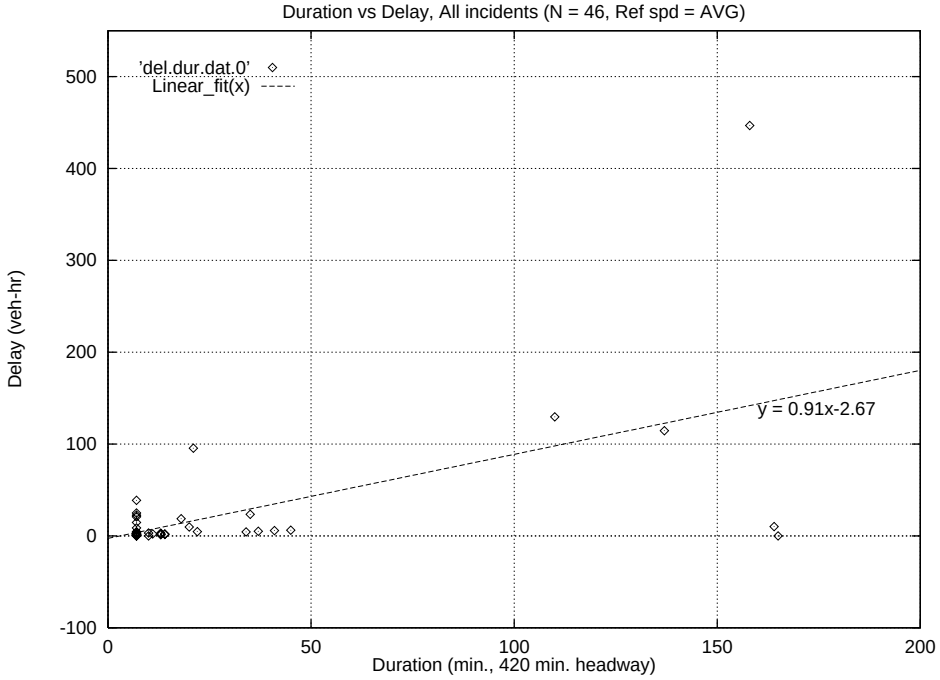

Figure 16.6: Incident Delay Versus Duration.

- The incidents that are placed on the correlation plots are only the ones that make it through the incident filter. Therefore you should generate an incident filter that will allow everything to pass through.

There is only one correlation plot for each shift of car data so the naming scheme is pretty straight forward. There are two main files that generated by the correlation routine that are named `incidentcor.gp` and `incidentcor.gv`. Both of these files generate a correlation plot but the first one dumps it to the printer and the second one displays it on the screen. The way to remember this is to think of the file extension as "Gnuplot Print" or "Gnuplot View." Although the only files the normal user would have to deal with are the two files mentioned above, there are a few different files that the correlation routine generates that are placed in the car shift directories. The naming scheme of these auxiliary files is as follows:

carX.idt : These files hold the key presses that the drivers made when they passed by an incident. The "X" stands for the car number.

dYY.b : These files hold the incident locations. The "YY" stands for the incident number - there is one file for each incident. These files have such short filenames due to a restriction in **gnuplot** on the length of the command line. There is nothing that can be done about this.

So a typical shift directory might look something like this:

```
car1/           d10.b           d19.b           d3.b
```

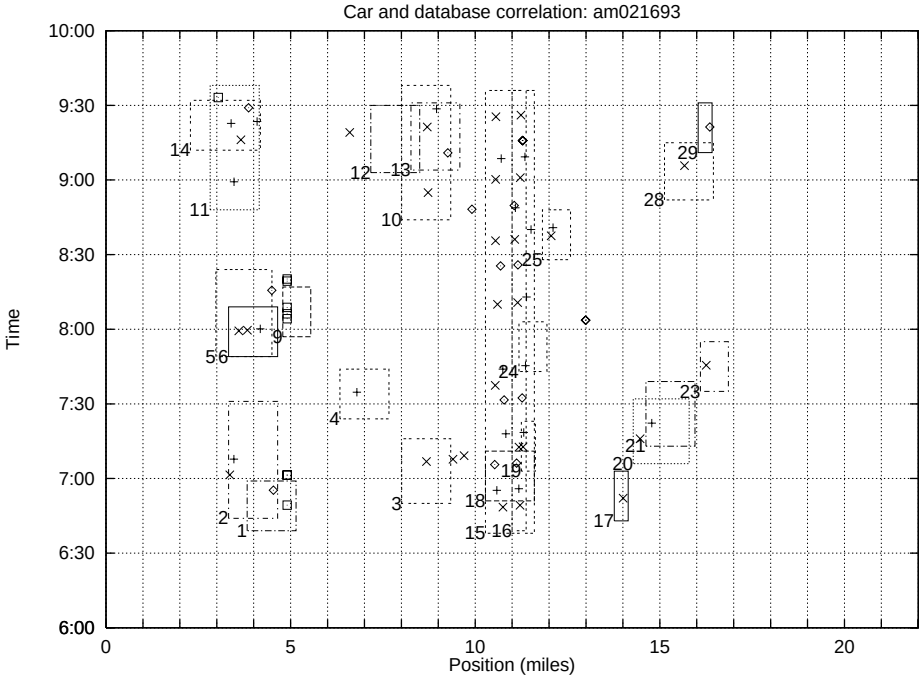

Figure 16.7: Incident Correlation Plot.

`car1.idt`	`d11.b`	`d2.b`	`d4.b`
`car2/`	`d12.b`	`d20.b`	`d5.b`
`car2.idt`	`d13.b`	`d21.b`	`d6.b`
`car3/`	`d14.b`	`d23.b`	`d9.b`
`car3.idt`	`d15.b`	`d24.b`	`incidentcor.gp`
`car4/`	`d16.b`	`d25.b`	`incidentcor.gv`
`car4.idt`	`d17.b`	`d28.b`	
`d1.b`	`d18.b`	`d29.b`	

As you can see, there are quite a few files with names of the form dYY.b and hence quite a few incidents. Note that I have left out the all of the travel time files that are normally found in this directory. All of the correlation files are stored under the car output directory in the individual shift directories.

16.3.3 The Contour Plots

When told to do so, the **fsp** program will generate contour plots of the loop data with the incidents placed on them. One important use of the contour plots is that they give you an easy way to figure out where the end of the queue is that built up behind an accident. This is important if you are attempting to figure out the incident space-time boxes for the delay calculations. An example of doing this is given in Section 12.11. The program will make three different types of contour plots:

Delay : These are plots where the contour lines are delay, which is in vehicle-hours. The root file name for these is `contour`. I know that it probably should have been `delay` but it

just didn't turn out that way.

Density : These are plots where the contour lines are density, which is in vehicles per mile. The root file name for these is `density1`.

Differential density : These are plots where the contour lines are the percentage over the average density. So if the density of traffic for one day at one spot was 200 vehicles/mile and the average density for that spot was 100 vehicles/mile then the differential density would be 1, which means 100% over the average. So the differential density are percentages. The root file name for these is `density2`.

For each type of contour plot the program will generate a plot for each direction and for each shift, for a total of 4 different plots for each day. The filenames for the gnuplot executable files that generate the contour plots have two parts: a root file name and an extension. The root file name tells what the contour lines are (either delay, density or differential density) and the extension specifies the direction, time, and output device. The naming convention for the gnuplot executable files is given below:

```
contour.g{n,s}{a,p}{p,v}
density1.g{n,s}{a,p}{p,v}
density2.g{n,s}{a,p}{p,v}
```

As was discussed above, the individual plot types each have their own root file name, but the extensions all have the same format. The brackets in the extensions above each represent one character that can be chosen out of the two characters inside the brackets. Something like `{n,s}`, which represents the second character in the extension, means that you can only choose the "n" or the "s" but not both. So the naming scheme is as follows: the second character in the extension signifies the northbound or southbound direction, the third character signifies the AM or PM shift, and the fourth character signifies whether to print the plot to a printer or to view it on the screen. For example, to view the delay contour plot for the northbound, AM shift you would use the gnuplot executable file `contour.gnav`. To print out the differential density plot of the southbound, evening shift you would use the gnuplot executable file `density2.gspp`.

There are also a couple of data files that the gnuplot executable files read in when making the plots. The naming scheme for these files is similar to the gnuplot executable files:

```
contour.{n,s}{a,p}
density1.{n,s}{a,p}
density2.{n,s}{a,p}
```

The first character in the extension signifies the direction, and the second character signifies the shift. Since these files hold the raw data, they were not meant to be read or processed by normal users - they are only read in by the gnuplot executable files.

There are couple of runfile parameters that need to be set in order to generate the contour plots:

- Since the contour plot generation routine is part of the incident processing section you need to tell the **fsp** program to process the incidents. This is done by setting the runfile parameter `PROCESS_INCIDENTS` to `YES_PROC_INCIDENTS` .

- The parameter `INC_CONTOUR_DELAY_PLOT` controls whether the contour plots are generated or not. This needs to be set to `YES_INC_CONTOUR_DELAY_PLOTS`.
- The incidents that are placed on the correlation plots are only the ones that make it through the incident filter. Therefore you should make an incident filter to your liking. One suggestion is to just filter out the accidents and see what kind of congestion they cause.

The contour plots are all placed in the individual loop day output directories. In Section 12.8 there is an example that shows how to generate the contour plots. Note that you first need to generate the loop averages before you can make the contour plots. Figures 16.8 through 16.10 give examples of the various contour plots that the program can generate.

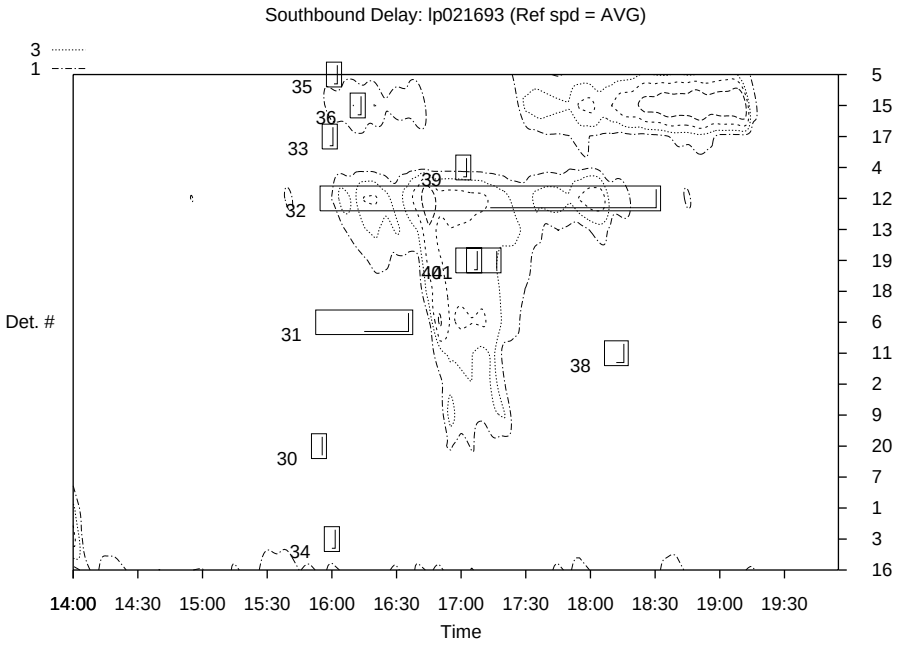

Figure 16.8: Contour Plot Of Delay.

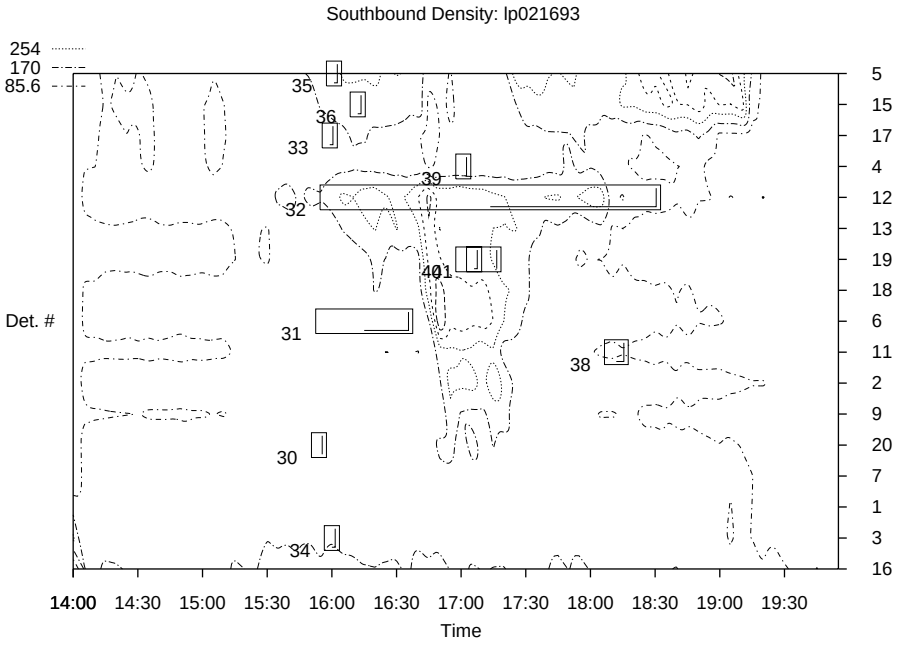

Figure 16.9: Contour Plot Of Density.

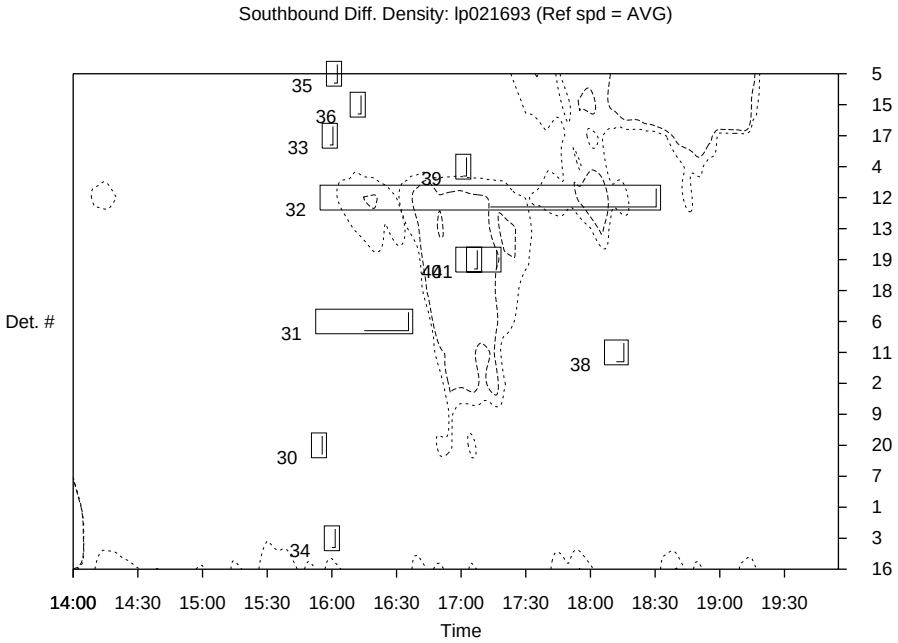

Figure 16.10: Contour Plot Of Differential Density.

Chapter 17

A Larger Picture: The Whole FSP Data Flow

Up to this point I have talked almost exclusively about how to run the **fsp** once the data is on the hard drive. Well, due to the large size of the data set, this begs the question, "How did the data get on the hard drive in the first place?" In this chapter I will try answer this question and to also give a slightly larger picture of the whole process than what I have done previously.

The data started off being generated in the test cars and the loop detectors. There were four probe cars that drove up and down the freeway for three hours in the morning and three hours in the afternoon. During that time period the loop detectors were also taking data. All of this data was stored on $3\frac{1}{2}$ inch floppy disks. Every few days the data from the cars and the loop detectors was collected and brought to the lab where they were transferred to the Sun Workstation. All of this is represented in Figure 17.1.

As you can see, the data started with the cars and the loop detectors which are underneath label 1 in this picture. The transfer of the data was done with the help of a PC and a device called a disk loader. A disk loader is an add on device to the PC that allows you to stack a bunch of disks in a hopper and then to automatically feed them into a disk drive one by one. A software program runs on the PC to control the transfer of the data from the floppy disks to the Sun. This program, called **ftran**, controls the disk loader, stores the data on the PC hard drive temporarily - step 3 in Figure 17.1 - and then transfers the data to the Sun across the local ethernet in step 4.

The **ftran** program actually does quite a bit of setup on the Sun every time that it runs. The first thing that it does is to create the directory structure that the **fsp** program is expecting. It then places the data in these directories. Next, it makes a runfile that corresponds to this data and places it in the runfile directory. It then makes a log file that lists out the what data was just transferred and places this in the log file directory. Finally, it makes a special script file and places it in the home directory of the main user. When run, this special script file will execute the **fsp** program with the recently generated runfile and will print out the error reports to the local printer. Once the analysis on the data has been run, the data can be transferred to a magnetic tape for easy storage. Currently, all of the data for an entire month can be stored on one 8mm tape. Since an 8mm tape costs about $9, this is a very cost effective way to store the data if you are done with it.

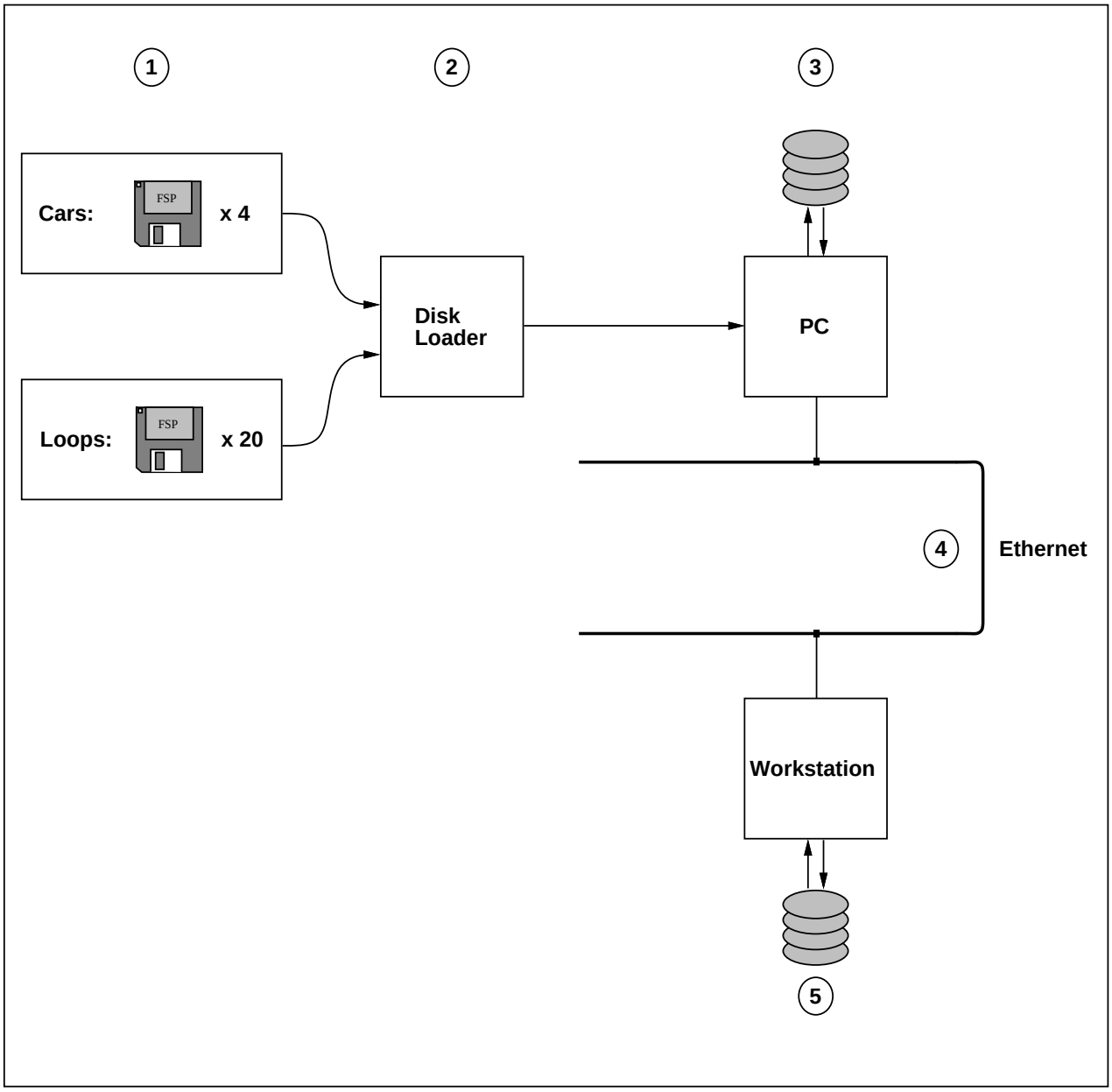

Figure 17.1: The Larger Picture.

Just to sum up what the **ftran** program does:

1. Control the disk loader such that it can read in all of the disks.
2. Figure out what data is present on the disks.
3. Make a runfile that corresponds to the data.
4. Make a log file that corresponds to the data.
5. Log onto the Sun.
6. Create the appropriate directory structure on the Sun.
7. Place the car and loop data in these directories.
8. Transfer the new runfile and log file.
9. Create a script file to run the **fsp** program with the new runfile and to print the error reports out to the printer.
10. Transfer the special script file to the home directory of the main user.
11. Log off the Sun and delete the data from the PC hard drive.

The program was designed this way so that somebody that was not very familiar with the **fsp** program could analyze the data. In order to run through the whole process, from the disks to the printouts of the error reports, they would only have to do a few tasks:

1. Place the disks in the disk loader.
2. Run the **ftran** program on the PC.
3. Log onto the Sun and run the special script file.
4. Pick up the printouts.

The slowest part in this whole process was the reading of the floppy disks by the disk loader. What was envisioned was that we could stack a bunch of disks in the disk loader and then go out to lunch. When we came back, all of the data would be transferred to the Sun waiting for us to run the analysis program. At the point, this process works pretty well.

One final point should be made about the way the **ftran** program names the runfiles that it generates and transfers to the Sun. Since file names on a PC can only be 8 characters long I had to come up with a naming scheme that had only 8 characters. What I came up with was the following format: rfXXYYZ.run. Where "XX" stands for the month, "YY" stands for the day, and "Z" stands for the transfer number of that day. The **ftran** program keeps track of the transfers that it does and it marks each one with a specific number for that day. This needed to be done so that the file name wasn't dependent on just the date. If it was and somebody transferred two batches of disks to the Sun then there is a possibility for confusion in what to name the files. For example, if there were 5 batches of disks that were transferred to the Sun on March 10th, then the runfiles would be named:

```
rf03101.run
rf03102.run
rf03103.run
rf03104.run
rf03105.run
```

Appendix A

Frequently Asked Questions and Warnings

This appendix contains a list of frequently asked questions and warnings about the **fsp** program. These were collected by having a few people around the country beta test the program. The points noted here are mainly from the questions that they asked. Since most people just want to work with the loop data, most of the information deals with that.

Note that these questions only deal with the current version of the software which is **fsp 1.1**. If you are using an earlier version, you might want to first glance at the changes that have been made. They are listed out in Appendix B.

A.1 General

Pointer: *Before downloading the support software for the* **xfsp** *program, you should check to see if it is already on your system.*

The **xfsp** program was written in a scripting language called Expect. In order for you to run the **xfsp** program you need to have the program **expectk** on your system. You can download the Expect software from the ftp server at UC Berkeley and install it yourself, but you should first check to see if it is installed on your machine. It is a very common program and most system administrators will have already installed this program. To see if you have the program **expectk** try the command: `which expectk`. This will tell you if you have it or not. Chapter 3 has instructions on how to download and install the program if you don't already have it.

Question: *Can I process data from both the before study and the after study at the same time?*

No, you can't. The **fsp** program can only process one type of data at a time. You can't work with two different data sets in the same runfile. You have to split up the work into two runfiles.

Question: *Can I ask the author any questions?*

Of course. I'd be happy to answer any questions that you might have. You can just send me email at `pettyk@eecs.berkeley.edu`.

A.2 The Loop Data

Pointer: *Download the processed loop data instead of generating it yourself.*

You should always try to download the processed loop data and use that. Learning to use the **fsp** program can be a pain. There are usually a couple of different processed data sets on the ftp server in the directory `/pub/PATH/FSP/Data/Processed`.

Question: *What is the format of the loop data?*

The loop data files are in two columns. The first column is the number of seconds since midnight and the second column is the value - either speed, occupancy, flow, delay, etc. You should look at the warnings about the units of the loop data files to make sure that you realize what each file contains.

Warning: *The units for the loop flow data files are not consistent.*

The units for the various loop data flow files are all different. The units for the individual lane flow files are in vehicles per time period. So if your output time period is 60 seconds, then the units are the number of vehicles per 60 seconds. An individual lane file is a loop output file that ends with a number. So the file `floop3.nc1` is the file that contains the counts, or flow, at detector #3 in the northbound direction for lane #1. A more thorough explanation of the loop file naming scheme is given in Section 15.3.

The units for the loop flow data that has been averaged over all of the lanes is in vehicles per lane per hour. These files all have extensions that end with the letter "d". The units for the on and off ramps are in vehicles per time period.

Question: *I made the individual lane files but there are lots of holes in the data. Why is this? Can this be fixed?*

The short answer is "no." The individual lane files have holes in them due to the 170 loop controllers being buggy. The program won't do anything to fix these holes. What the program does fix is the data that is aggregated over all of the lanes. There is an extended discussion of this point in Section 11.2.

Question: *I created the loop data speed files and I noticed that at some times the speed was 409 (miles/hour). How could that happen?*

The loop detector speed is calculated by figuring out the amount of time that it takes for a vehicle to pass over both loop detectors. What the program does is it looks at the time difference in the falling edges of the pulses generated by a vehicle moving over the loop detectors. You should refer to Figure 4.5 and the discussion there if you don't know how speeds are calculated. The key thing is that the vehicle needs to be picked up by both detectors.

There are couple of things that could happen that could cause the loop speeds to be incorrect:

- If a truck goes over the loop detector, the loop detector might pick up each axle as a separate vehicle. This is probably the most likely cause of incorrect speeds.

- If a car is switching lanes and it is picked up by the first detector but not the second then this could mess up the speeds.
- On the other hand, if a car is switching into the lane then it's possible that the second detector will pick up the car but not the first.
- The thresholds in the loop detectors could be calibrated incorrectly. This would be more likely to cause the occupancy reading to be incorrect than the speed to wrong, though.

The best way around this is to use data of a higher output time period. This will cause the spurious spikes to be averaged out.

Warning: *The speed for the loop files averaged over all of the lanes is not the normal average.*

The speed in the loop output speed files is the weighted harmonic mean speed. This is slightly different from the mean average speed. The mean average speed would be something like:

$$V_m = \frac{\sum_{i=1}^{N} F_i V_i}{\sum_{i=1}^{N} F_i},$$

and the weighted harmonic mean speed is:

$$V_h = \frac{\sum_{i=1}^{N} F_i}{\sum_{i=1}^{N} \frac{F_i}{V_i}}.$$

Where F_i is the flow for lane i of the freeway, V is the speed for lane i, and N is the number of lanes. You should refer to Section 11.1 to see why the speed is calculated this way.

Question: *I generated the loop speeds for a 1 second output period and there are a lot of values that are zero. Then I tried to filter this data with the exponential filter and the average wasn't correct. Why is this?*

What's going on here is that whenever you are filtering the loop data and there is a zero speed on the freeway the fsp program just sticks in the last speed. So if there is a whole series of zero speeds, then there could be a whole series of values that are the same.

This was done because if the data aggregation period is small then in the early morning periods there were times when there wouldn't be any cars travelling over the detectors and the fsp program was simply putting zeros in those spots. When we went to filter this data (with our exponential filter) all of those zeros were causing the filtered speed data to be artificially low. Hence our delay calculations were completely off. The way that we fixed this is to have the speed at a detector be the last speed when no cars went over the detector.

The reason that this is ok, for our purposes, is because we always want to get the delay or the density. And this involves multiplying the speed by the flow. So whenever no cars have past over the detector, and hence the flow is zero, the speed doesn't matter.

If you need to have the speed data be zero when there are no cars that travel over the detector then just don't filter the data and this feature will be turned off. So if you generate the loop data with a 1 second output period without filtering then there will be a lot of time periods that have zero speeds.

If you want the program to carry speeds forward but you don't want to filter the data then you can specify a filter value of 0.01. This is such a small filter value that the effect on the data will be negligible.

Warning: *The loop data speed value filter is reset in the morning.*

If the loop speed starts off at zero, which is very common in the early morning, then the filter is reset the first time it changes from zero. This way the filter doesn't get "stuck" at a low number. So if you are filtering the loop data with a large filter value you may notice that for the morning period there is a jump in the speed, that shouldn't happen in a true filter, when it changes from zero to some positive value.

Warning: *Although the data for the PM period starts at 2:00 pm, the first data point for the PM period never includes 2:00 pm.*

If you are generating the loop data with an output period of 5 minutes, the loop data reported for some time is the data corresponding to the previous 5 minutes. The program simply sums and/or averages the data over the previous 5 minutes and reports it to you. Since there is a skip from 10am until 2pm the 5 minutes previous to 2pm are not in the raw loop data. Therefore, the program has to wait until 2:05 to make its first report.

Warning: *The AM data is missing the first second of data.*

The first line of each loop data file is read in and used to setup the system (things like configure the output, initialize the variable structures). This line of data is then discarded. The program then starts reading in the data for real. Since each line of raw loop data corresponds to a second of data, the first second is always missing. The problem arises when the program tries to get an average for the different variables. The program thinks that the data was collected for an entire time period, not for a time period minus one second. Therefore, the data for the first time period will be slightly off. It might even be fractional.

Warning: *The loop data file extensions are longer than three characters.*

The file extensions are too long for a PC. There are files that have file name extensions of more than three characters, like `floop4.ss10`. If you try to untar these files then they will be saved with only three characters in the extension. This means that the file `floop4.ss10` will be saved as `floop5.ss1`. This is unfortunate because this will overwrite the proper file by that name.

If you need to have these files then what you need to do is to extract the troublesome files one at a time and then move them to a safe place. This can be done by specifying to the tar program which files you want to extract. For example, if you were trying to extract the files `floop13.sc1` and `floop13.sc10` from the tar file `lp021693.tar` then the steps would look something like this:

```
pettyk 1: ls
lp021693/       lp021693.tar
pettyk 2: tar xvf lp021693.tar lp021693/floop13.sc1
x lp021693/floop13.sc1, 554287 bytes, 1083 tape blocks
pettyk 3: ls
lp021693/       lp021693.tar
pettyk 4: ls lp021693
floop13.sc1
pettyk 5: mv lp021693/floop13.sc1 lp021693/lp13sc1.dat
pettyk 6: tar xvf lp021693.tar lp021693/floop13.sc10
x lp021693/floop13.sc10, 554293 bytes, 1083 tape blocks
pettyk 7: ls lp021693
floop13.sc1    lp13sc1.dat
pettyk 8: mv lp021693/floop13.sc1 lp021693/lp13sc10.dat
pettyk 9: ls lp021693
flp13sc1.dat    lp13sc10.dat
```

As you can see above, I first extracted the file `floop13.sc1` from the tar file and then I moved it to a different file: `lp13sc1.dat`. Then, I extracted the file `floop13.sc10` from the tar file. Note that this file was saved as `floop13.sc1` because of the file extension limitation. I then moved this file to `lp13sc10.dat`.

Question: *Can I use the* **xfsp** *program without downloading all of the raw data?*

The answer is "Yes" but the explanation is rather long - so bear with me.

Before I answer the question, I think that I should point out why it is being asked. What this person wanted to do was to find the delays for various incidents but they didn't have enough disk space to download the raw loop data. So they wanted to know if it would be possible to put only the processed data on their machine and then use the **xfsp** program to calculate the delays.

If you are just using the **fsp** program without the **xfsp** program the the answer to this is "yes" because when the **fsp** program is calculating the delay for an incident, it doesn't need to have the raw loop data files on the system - only the processed loop data. To do this you simply specify in the runfile that you don't want to regenerate the processed data from the raw data. The **fsp** program will notice this in the runfile and it won't look for the raw loop data files. It will only look for the processed files because that's what it needs to calculate the delay for each incident. So if you were using only the **fsp** program then you would have to download only the processed data and not the raw loop data.

But a problem shows up when you try to do this with the **xfsp** program. One of the screens of the **xfsp** program presents the user with a list of the raw loop detector data that is available. The program gets this list by searching through the loop data directories for loop data files of the form `loop3.dat` or `loop3.dat.Z`. For each file that it finds in that format it makes a button in the loop data select window that you can set to indicate whether to use that data or not. The problem with not downloading the raw loop data is that the **xfsp** program will find the loop directories but it won't find the raw loop data files inside of them. Therefore it will get confused and say something bad.

But you can fool the **xfsp** program into thinking that it has the raw loop data files when it fact it doesn't. To do this you need to create a "stub file" for each possible raw loop data file. A stub file is a file that has the same name as a raw loop data file, but the file itself is empty. This way, when you click on the "Loop Data Select" button the **xfsp** program will see these files and mistakenly think that it has them (the program will not check the size of the files and see that they have nothing inside them). You can download the stubs from the anonymous ftp server at UC Berkeley. See the discussion in Section 2.1 if you don't know how to download stuff using ftp. The files are in the following locations:

```
/pub/PATH/FSP/Misc/lp.stub.set1.tar.Z
/pub/PATH/FSP/Misc/lp.stub.set2.tar.Z
```

To successfully fool the **xfsp** program, you will need to do a couple of things:

1. Grab one or both of the stub files.
2. Uncompress and untar this file. You will get a directory called Loopdata (make sure that you untar this in a place that doesn't already have a "Loopdata" directory).
3. When you run the **xfsp** program specify the directory that you just made as the "Loop data directory". You should be able to open up the "loop data select" button and see all of the loop files.
4. Place the processed loop data files that you downloaded under an output directory. You should probably change the permissions on these files to make then un-writable. The reason for this is that there are options in the fsp program to delete certain output files and you wouldn't want to lose them accidentally.
5. When you run the **xfsp** program specify the output directory to the proper directory.

Your final directory structure should look something like this:

```
A) /home/FSP/Set1/Input/Loopdata
B) /home/FSP/Set1/Input/Cardata
C) /home/FSP/Set1/Input/Incdata

D) /home/FSP/Set1/Output
E) /home/FSP/Set1/Output/Loopdata
F) /home/FSP/Set1/Output/Loopdata/lp031093
G) /home/FSP/Set1/Output/Loopdata/lp031193
H) /home/FSP/Set1/Output/Loopdata/lp031293
I) /home/FSP/Set1/Output/Loopdata/lp031393
J)                                      .
```

So your input loop directory is (A). This is where you should place the stub directories and files. The main output directory is (D). You should place the processed loop data files that you downloaded from UC Berkeley in directory (E). Examples of these processed loop file directories are (F) - (J). Note that you don't need to create the directories (B) or (C). You only need these if you want to do something with the car data or the incident data.

A.3 The Probe Vehicle Data

Question: *What is the probe data used for?*

Unfortunately, the probe data was not very reliable. There are long discussions in Sections 5.1 and 5.3 discussing the problems with the probe vehicle data. In summary I will say that we didn't use the probe vehicles for anything in our study other than generating the incident logs.

Question: *Sometime when I generate the incident pass times (the times when a probe went passed an incident and didn't see it) the time are the same as the time that the incident was first witnessed.*

This can happen if the first probe vehicle at the start of the shift witnessed the incident. In this case, there were no probe vehicles that went down the freeway before the first one. Therefore, there wasn't a vehicle that drove by the incident location and didn't see the incident. In this case, the program simply says that the before pass time is the same as the time that the incident was first witnessed. You should see Section 16.2.4 for a discussion of the incident pass times.

Question: *What exactly are the distances travelled by the probe vehicles?*

The distance travelled down the freeway by the probe vehicles can be broken down into four different components:

Southbound travel distance: This is the distance on the freeway that the vehicle travels in the southbound direction.

Southern turn-around distance: This is the distance at the southern end of the freeway that the vehicle needs to travel to get back on the freeway heading northbound. It is the distance of the southbound off ramp, the freeway overpass, and the northbound on ramp.

Northbound travel distance: This is the distance on the freeway that the vehicle travels in the northbound direction.

Northern turn-around distance: This is the same as the souther turn-around distance but in reverse.

So the equation for the total distance traveled per loop would look like:

$$\begin{aligned} D_{tot} &= [\text{Southbound distance}] + [\text{Southern turn around}] \\ &\quad + [\text{Northbound distance}] + [\text{Northern turn around}] \end{aligned}$$

The values given in Table A.3 are the starting and ending distances of the parameters and they are in feet. Note that the values for the southbound and northbound travel distances are the entire length of the freeway - they are not the distance over which we have loop detector data. Also note that we don't have a lot of confidence in these numbers.

If you are looking for these distances in the car run files then you might not find them. Some of the reasons that these files might be messed up are as follows:

Parameter	Start	End	Length
Southbound travel distance	0	47600	47600
Southern turn around	47600	49870	2270
Northbound travel distance	49870	97620	47750
Northern turn around	97620	≈100000	≈2300

Table A.1: Travel Distances (in feet).

1. If the driver didn't press the start key during one of their runs then there will be two runs pushed together. This is a very common mistake - but it is easy to spot because the file will be twice as long.
2. The vehicles didn't always make a loop when they were driving during the study period. Sometimes, if they arrived too early, they were told to drive to the Denny's by the entrance of the freeway. Since this data is stored at the end of the file (because the program doesn't know that the run is over), the total length of each run might vary.
3. There are many driver errors. Sometimes the drivers forgot to press the start keys at the start of the run and instead pressed them once they were down the road a ways. The program of course assumes that when the start keys were pressed that the car was at one specific location. What this means is that the car distance file for this run will be incorrect. This is a very hard error to spot.
4. I thought that the turn around point in the run had changed during the course of the experiment. But it seems that the distance that that change was not significant.

A more thorough discussion of the probe vehicle data is given in Section 4.3 and in Chapter 14.

A.4 The Incident Database

Question: *Do I have to specify an incident filter? What if I don't want to filter any incidents?*

The incident filter always has to be specified on the command line. If you don't want to do anything with the incidents then you need to set the following runfile parameter:

```
PROCESS_INCIDENTS = NO_PROC_INCIDENTS
```

This will tell the program to not filter any incidents. If you want to make a dummy incident filter file you can create a file with the following lines:

```
#DATA_TYPE              = F
#INC_NUMBER             =
```

Actually, you can have any size file as long as the first characters in all the lines are #'s (which is the comment character for the incident filter).

Question: *How do I figure out the tow truck assisted incidents?*

In order to know if a tow truck showed up we can just check to see if a tow truck arrival time is listed. If it is then we are assured that a tow truck arrived. So in the incident filter, you should have the following line:

```
TOW_ARRIVAL              = 6:00 - 20:00
```

Since we only have probe vehicle data from 6:30am until 9:30am and then again from 3:30pm until 6:30pm, the time period from 6:00am until 8pm should cover all of the tow truck arrival times. You should look at the example given in Section 9.3.4.

Question: *How do I find out how many incidents occurred at some section of the road during some time period?*

You can do this by specifying the correct incident filter. The best thing to do is to review the examples presented in Section 9.3. They talk about the way to filter different stuff out of the incident database. A terse list of what to do is given below:

1. Look at the table in Section 4.5 that discusses the incident database and figure out the incident database fields that you need to filter. To filter out the incidents for a certain section of the road and a certain time period, then you want the fields:

```
Column  Name             Description
E       Time:            Time listed in military time
I       Link Identity:   Link identity according to between exits
```

2. Look at Tables 9.1 and/or 9.2 to learn what the parameters are in the incident filter that correspond to these fields. In this case they are:

```
Column  Name             Field Descriptor
E       Time             TIME
I       Link Identity    LINK_ID
```

3. Specify the desired quantities for these parameters in the incident filter. For example, if you want incidents that occurred on the link between A-Street and Winton from 2pm until 4pm you would put the following lines in the incident filter:

```
TIME             = 14:00 - 16:00
LINK_ID          = 4
```

4. Run the **fsp** program with this incident filter and a runfile that processes the incidents.

Question: *Is the time recorded in the "time" field of the incident database when an operator discovered it, when police, etc reported it, or what has been determined to be the start of the incident?*

The time field in the incident database is the time that the first probe vehicle witnessed the incident. Note that since there were only four probe vehicles, with an average headway time between them of seven minutes, the starting and ending times of the incidents are sampled by seven minutes. One part of the **fsp** program attempted to fix this but it didn't quite work. See the discussion in Sections 5.3.2 and 16.2.4 for more information.

Question: *What does the DATA_TYPE field mean in the incident filter?*

The DATA_TYPE descriptor was meant to distinguish between incidents from different sources. The "F" meant that the incident was from our field data, the "C" meant that the incident was from the CADD database, and the "T" meant something that I can't remember. As it turns out, we never included the incidents from any source other than our field data. As a result, you don't need to filter on this descriptor because every incident in the database has this as an "F".

Appendix B

Changes From Version 1.0 to 1.1

This appendix contains a list bugs that were fixed, and changes that were made, since the first release of the **fsp** software. They are more or less in descending order of importance.

- The **xfsp** program no longer uses the extended version of Tcl or BLT. These program were extensions to the standard Tcl/Tk program that were needed to control the running of the **fsp** program from the **xfsp** program. Now the **xfsp** program uses the standard program Expect which should already be installed on most systems. This should reduce the amount of time it takes to install the **xfsp** program considerably.

- The **fsp** program no longer has a short and a long data set. There is only a short data set and it is only referred to as the data set. I realized that there was no reason to have a long data set because all of the subroutines only worked with the short data set. Note that this means that the runfile parameters have changed. Although you can still use the old runfile parameters, I would encourage you to use the new ones.

- The correction factors for the loop data were put in the code. You can now fix the loop data when there are consistency errors. Quite a bit of code was changed in `log_fsp.c` to accommodate this. The files that this generates are the `hloop` files. For a more thorough discussion of the consistency correction see Sections 5.2.2 and 11.2.3.

- If the duration of the incident is zero and there is no user-added headway then the delay is automatically zero. The program used to sometimes pull in the delay for that one time period but now it doesn't. This was changed in the file `inc_util.c`.

- The program now spits out the processed loop data files with only one digit past the decimal place. This should reduce the amount of space needed to store the data by $\frac{1}{3}$ in some cases!!!

- The program used to carry the loop speed over when the current speed was zero. This was only done because we needed to filter the speed and all of those zeros were pulling down the loop speed. Now, if the loop filtering factor is zero, the speeds are not carried over.

- In the file `fsp_calc.c` I changed an error in fprintf() where I wasn't passing the file name.

- In the file `fsp_calc.c` I fixed the calculation of the interpolated speed so that it checks for a zero before doing the division. What was happening before is that I wasn't checking to make sure that the divisor wasn't zero before doing division.
- I increased the number of holes in the loop data that you can fill. This was defined in the parameter MAX_NUM_LOOP_HOLES in `fsp.h`. If you were generating the data for a very small output period then it was possible for the table used to hold the holes to run out of room and cause the program to crash.
- There was a problem with the first entry after the start of the pm time period. I had to redefine the start of the PM period to be 50401 instead of 50400, reset the data structures once we got to the PM period, and reset the loop filter. The result is that there is one less value in the PM study now.

Bibliography

[1] John K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, 1993.

[2] Don Libes. *Exploring Expect: A Tcl-based Tookit for Automating Interactive Applications.* O'Reilly and Associates, Inc., 1994.

[3] A. Skabardonis, H. Noeimi, K. Petty, D. Rydzewski, P. P. Varaiya, H. Al-Deek. Freeway Service Patrols Evaluation. Technical Report UCB-ITS-PRR-95-5, Institute of Transportation Studies, University of California, Berkeley, June 1994.

Index