**Chapter 6**

**METHODOLOGY**

**6.1 Algorithms**

**6.1.1 Algorithm 1: Getting snapshots of junctions from traffic layer provided by google maps**

Step 1: Start.

Step 2: Fetch the Lat/Long from the pre built data base.

Step 3: Construct a map canvas around the Lat/Long.

Step 4: Call the headless browser to navigate to the constructed map canvas.

Step 5: Wait until the map is loaded into the headless browser.

Step 6: Take a snap shot of the loaded map.

Step 7: The snap shot in JPEG format with current date and time in to a hard coded location.

Step 8: Repeat Step 3 through step 7 for every min.

**6.1.2 Algorithm 2: Filtering the snapshots taken**

step1: Start

step2: Fetch the image from hard coded location.

Step3: With the help of OpenCV library, filter the color (red, green, yellow) in the given range.

Step4: Three binary images/filtered images in .png format are obtained.

Step5: These binary images are given as input for percentage calculation.

Step6: Repeat Step2 through step5 until all snapshots are fetched from database.

**6.1.3 Algorithm 3: Quantifying the data (percentage calculation)**

Step1: Start

Step2: Binary images of a particular snapshot are given as input for percentage calculation.

Step3: Using OpenCV libraries calculate number of black pixel count and white pixel count for each binary image.

Step4: Calculate white pixel percentage over total pixel count.

Step5: Percentage values are stored in database.

Step6: Stop.

**6.1.4 Algorithm 4: The Prediction Engine**

Step1: Start

Step2: The percentage values obtained for one full day of a junction is fitted with a polynomial using the GSL library.

Step3: Step2 is performed for every day of the week for at least 2 weeks

Step4: The average of all polynomials for a day in week with least deviation is considered for the prediction

Step5: The predicted data is evaluated with the actual data

Step6: Repeat the Step2 through Step5 until a desired level of accuracy is achieved

**6.1.5 Algorithm 5: The Display Engine**

Step1: Start

Step2: Obtain the date from the datepicker

Step3: Obtain the junction in query from the list view

Step4: Make an AJAX request to generate JSON data for the supplied date and the junction in query

Step5: The JSON data returned is passed to the Google Charts API to plot a Line chart for Time vs Traffic density

Step6: Step2 through Step5 is repeated for a variety of inputs.

**6.2 Code Description**

**6.2.1 Server side**

**6.2.1.1 Code 1**

**Name** : Main.java

**Description** :

* This is the main class, it contains the main method that starts/triggers all the activities.
* The main method creates and executes a thread for every junction.

**6.2.1.2 Code 2**

**Name** : GetSnap.java

**Description** :

* This class is responsible for capturing the snapshots of th junctions under observation.
* GetSnap class implements runnable.
* GetSnap( ) initializes a thread with the given name, cmd and folderPath.
* @param name the name of the thread to be created.
* @param cmd a string array of command's to be executed by this thread.
* @param folderPath the absolute path of the junction for which the thread is being created.
* run( ) method is overridden and it consists of all the tasks that this thread must Execute.
* start( ) method is used to create and start a new thread with the given name.

**6.2.1.3 Code 3**

**Name** : MoveFiles.java

**Description** :

* + This class is used to move the snapshots to their destination based on the custom hierarchy designed.
* MoveFiles( ) constructs and initializes a new file (folder) with the given folderPath.
* @param folderPath the absolute path of the junction being operated on.
* Move( ) method is used to move the files to destination.
* This class implements FilenameFilter interface.
* MyFileNameFilter( ) constructs and initializes a filter with the provided extension
* @param ext the extension upon which the files are filtered
* This method is overridden from the FilenameFilter interface
* @param dir the directory that contains files to which the filter has to be applied
* @param name the name of the file
* @return true if the extension is matched else false

**6.2.1.4 Code 4**

**Name** : FilterSnaps.java

**Description** :

* + This class applies various filters to a captured snapshot andreturns thefiltered images.
* FilterSnaps( ) constructs and initializes a new file (folder) with the given Folder Path.
* @param folderPath the absolute path of the junction being operated on.
* startFilter( ) method starts the process of applying filters on the captured Snapshot.
* This class implements FilenameFilter interface.
* MyFileNameFilter( ) constructs and initializes a filter with the provided extension
* @param ext the extension upon which the files are filtered.
* accept( ) method is overridden from the FilenameFilter interface.
* @param dir the directory that contains files to which the filter has to be applied.
* @param name the name of the file.
* @return true if the extension is matched else false.

**6.2.1.5 Code 5**

**Name** : FetchPercent.java

**Description** :

* This class returns the amount of color value in percent for a given filtered image
* getJunctionName( ) method returns the name of the junction
* @return the name of the junction as a string.
* getYear( ), getMonth( ), getDate( ), getHour( ), getMinute( ), getSecond( ) methods returns the year, month, date/day, hour, minute, second of the snapshot respectively.
* @return the year, month, date/day, hour, minute, second as strings respectively.
* getpRed( ), getpYellow( ), getpGreen( ) methods returns the amount of red value, yellow value and green value in percent of the snapshot respectively.
* @return the red value, yellow value, green value in percent as a float.
* FetchPercent( ) constructs and initializes a new file (folder) with the given folderPath.
* @param folderPath the absolute path of the junction being operated on.
* getPercent( ) method computes the percent value of a color from a snapshot.
* This class implements FilenameFilter interface.
* MyFileNameFilter( ) constructs and initializes the extension and filename length to the filter.
* @param ext the extension upon which the files are filtered.
* @param fileNameLength the length of the filename upon which the files are filtered.
* accept( ) method is overridden from the FilenameFilter interface.
* @param dir the directory that contains files to which the filter has to be applied.
* @param name the name of the file.
* @return true if the length of filename is matched else false.

**6.2.1.6 Code 6**

**Name** : DBConnectionConfiguration.java

**Description** :

* This class is used to establish a connection to the database
* getConnection( ) method creates a connection to the remote database using the mysql driver for JAVA.
* @return the connection instance to the remote database

**6.2.1.7 Code 7**

**Name** : CreateDBObject.java

**Description** :

* This class creates an object which represents all the necessary information about a given snapshot
* getTableName( ) method returns the name of the table in database which represents the junction of the snapshot.
* @return the name of the table as a string.
* setTableName( ) method is used to set the value of table name based on junction name.
* @param junctionName the name of junction as a string.
* CreateDBObject( ) constructs and initializes a DBObject with given junctionName, year, month, date, hour, minute, second, millisecond, pRed, pYellow, pGreen.
* @param junctionName the name of the junction as a string.
* @param year the year of the snapshot as a string.
* @param month the month of the snapshot as a string.
* @param date the date of the snapshot as a string.
* @param hour the hour of the snapshot as a string.
* @param minute the minute of the snapshot as a string.
* @param second the second of the snapshot as a string.
* @param millisecond the millisecond of the snapshot as a string.
* @param pRed the red value in percent as a float.
* @param pYellow the yellow value in percent as a float.
* @param pGreen the green value in percent as a float.
* getter and setter methods written for year, month, date, hour, minute, second, millisecond, red value, yellow value and green value.

**6.2.1.8 Code 8**

**Name** : AddDBObject.java

**Description** :

* This class is used to add a DB object to a remote database
* insert( ) method is to insert a row into the remote database
* @param object the DB object to be inserted into the database

**6.2.1.9 Code 9**

**Name** : GetData.java

**Description** :

* This class is used to get data from the remote database.
* GetData( ) constructs and initializes the name of the database table.
* @param junctionName the name of the junction under operation.
* fetchData( ) method returns all the records that match the query as DB objects.
* @return list of DB objects that matched the query.

**6.2.2 Client side**

**6.2.2.1 Code 1**

**Name** : poly.cpp

**Input(s)** : The file containing traffic density values (Y-axis file).

**Output(s)** : The predicted (Polynomial fitted traffic density) values in a file.

**Description** : Given the files mentioned in input, the program fits a polynomial of specified degree and returns the Y values for the corresponding X values (minPlot file values).

**Argc** : 3

**Args** : The file containing traffic density values (Y-axis file).

// The following example shows how to fit a polynomial of degree 6

**Example** : ./poly 2016-04-08-silkboard-red.txt 2016-04-08-minPlot.txt 6 > 2016-04-08-silkboard-red-predicted.txt

**6.2.2.2 Code 2**

**Name**  : polyWithCoEff.cpp

**Input(s)** : The file containing traffic density values (Y-axis file). The corresponding minPlot file, the name of the coEff file.

**Output(s)** : The predicted (Polynomial fitted traffic density values) values in a file & the fitted polynomial co-efficients in another file.

**Description** : Given the files mentioned in input, the program fits a polynomial of degree 6 and returns the Y values for the corresponding X values (minPlot file values). It also generates a file with the name provided as 3rd argument to the program, which contains the value of coEfficients of the polynomial that was fitted for the data. Note: It is required to redirect the output using the redirect operator " > " into a file according to the naming principles. An example is shown for the same below.

**Argc**  : 3

**Args** : The file containing traffic density values (Y-axis file), The corresponding minPlot file, the name of the coEff file

**Example** : ./polyWithCoEff 2016-04-08-silkboard-red.txt 2016-04-08-minPlot.txt 2016-04-08-silkboard-red-coEff.txt > 2016-04-08 silkboard-red-predicted.txt

**6.2.2.3 Code 3**

**Name** : predictorPoly.cpp

**Input(s)** : standardDateTime.txt [contains values of time in seconds (0 - 86340) for one day with a difference of 60 seconds], coEff.txt [contains the coefficients of the polynomial]

**Output(s)** : Y [Traffic density] values for the given X data [standardDateTime.txt] using the coefficients for polynomial from coEff.txt

**Description** : Given the files mentioned in input, the program returns the Y values for the corresponding X values (standardDateTime.txt values).Note: It is required to redirect the output using the redirect operator " > " into a file according to the naming principles. An example is shown for the same below.

**Argc** : 2

**Args** : standardDateTime.txt, coEff.txt

**Example** : ./predictorPolystandardDateTime.txt coEff.txt >standardTraffic.txt

**6.2.2.4 Code 4**

**Name**  : StandardDateTime.cpp

**Input(s)** : --No Inputs--

**Output(s)** : The time values in seconds ranging from 0 - 86340 with a difference of 60 seconds.

**Description** : Generates time values in seconds ranging from 0 - 86340 with a difference of 60 seconds. Note: It is required to redirect the output using the redirect operator " > " into a file according to the naming principles. An example is shown for the same below.

**Argc** : 0

**Args** : --

**Example** : ./StandardDateTime > standardDateTime.txt

**6.3 Summary**

In this chapter, the algorithms used and brief explanation of the content of the codes are discussed.