Video Analytics Anomaly and Trend Detection

(VAAT)

Application Programming Interface

Version 0.3.3

October 23, 2014

Table of Contents

1 INTRODUCTION 4

1.1 SAV HIGH LEVEL DESCRIPTION 4
1.2 INTRODUCTION TO THE API 4
1.3 DOCUMENT ORGANIZATION 5

2 API DESCRIPTION 6

2.1 GENERAL API CHARACTERISTICS 6
2.2 RUN-TIME SUPPORT 6
2.3 CONFIGURATION FILES 6
2.3.1 General Description 6
2.3.2 Configurable Parameters 7
2.3.3 Usage Examples 7
2.4 INITIALIZATION FUNCTIONS 8
2.4.1 extractArgs 8
2.4.2 _vaatInitialize 8
2.4.3 _vaatInitializeEx 9
2.4.4 MarshallStrArray 10
2.4.5 MarshallStrArrayRev 10
2.4.6 _vaatInitSavStore 10
2.5 CONFIGURATION FUNCTIONS 11
2.5.1 _vaatGetConfigDbl 11
2.5.2 _vaatGetConfigInt 11
2.5.3 _vaatGetConfigStr 11
2.6 PROCESSING FUNCTIONS 12
2.6.1 _vaatSAVbar 12
2.6.2 _vaatTrendPrlUpdate 12
2.6.3 _vaatStoreSAVbar 12
2.6.4 _vaatClearSAV 12
2.6.5 _vaatGetPredominantFeature 13
2.6.6 _vaatListLevel 13
2.7 TERMINATION FUNCTIONS 13
2.7.1 _vaatTerminate 13
2.8 HELPER FUNCTIONS 14
2.8.1 _vaatGetVideoParams 14
2.8.2 _vaatDisplay 14
2.8.3 _vaatDisplayHelp 14
2.8.4 _vaatCheckBgReset 14
2.8.5 _vaatGetFrameRate 15
2.8.6 _vaatGetVideoIn 15
2.8.7 _vaatGetCurrentVideoFrame 15
2.8.8 _vaatGetHbitMap 15
2.8.9 _vaatGetCurrentVideoFrameAddr 15
2.8.10 _vaatStoreThruCurrentVideoFrameAddr 15
2.9 FILE MANAGEMENT FUNCTIONS 15
2.9.1 _vaatGetHandle 16
2.9.2 _vaatSAVfileOpen 16
2.9.3 _vaatSAVfilePrepare 16
2.9.4 _vaatGetSAV 16

2.9.5 _vaatRescaleSAV 16
2.9.6 _vaatGetPredominantFeatureBySAV 16
2.9.7 Example 17
2.9.8 Data Representation 17
2.10 ADVANCED FILE MANAGEMENT FUNCTIONS 18
2.10.1 _vaatCreateSAV 18
2.10.2 _vaatZeroSAV 18
2.10.3 _vaatSerializeSAV 18
2.10.4 _vaatGetPackerStyle 19
2.10.5 _vaatGetSAVfileAttr 19
2.10.6 _ vaatGetRawSAVavgFileHeader 19
2.10.7 _vaatGetSSfactor 20

3 TREND DATABASE 21

3.1 DATABASE ORGANIZATION 21
3.2 CONSOLIDATION OF TREND INFORMATION 21

4 SAMPLE APPLICATIONS 23

4.1 PROGRAM DISTRIBUTION 23
4.2 STARTUP 23
4.3 EXECUTION 23
4.4 VIDEO FILE SOURCE 24

5 ACTIVITY GRAPH 25

5.1 OPERATION 25
5.2 ALGORITHM 25

1 Introduction

Video Analytics Anomaly Trend detection software (VAAT) processes, stores, retrieves, and compares Scene Activity Vectors (SAV). This document describes the Application Programming Interface (API) for the VAAT software, release 0.2.1.

1.1 SAV High Level Description

An SAV describes motion within blocks of a video frame, where these blocks are organized in scale-space from level 0 (a single block comprising the full frame) to higher levels (typically up to level 5 or 6) of many smaller blocks. Each block is described by a vector of motion features: density, direction, velocity, color, and 4 deviation measurements of these 4 features respectively).

There are 3 time frames of SAV in VAAT: 1) SAV is computed every frame; 2) SAVbar is an average of SAVs, and computed each SAVbar period (1 sec. by default); and 3) trend is computed over a trend period (60 seconds by default) and stored for a particular time of the day (e.g., trend at 14:42 on Monday). For more information on SAV processing, see a technical paper associated with this documentation.

1.2 Introduction to the API

The VAAT programing API consists of six files used in different development environments as shown in the next table.

Module	Windows C++	Windows C#	Linux
vaatApi.dll	✓	✓	
vaatApi.lib	✓		
vaatApi.h	✓		✓
SAVparams.h	✓		✓
vaatApi.cs		✓	
libvaat.so.x.y.z			✓

Table 1 VAAT API Development Modules

C++ VAAT application source code must include **vaatApi.h**. **VaatApi.h** includes **SAVparams.h** therefore **SAVparams.h** need not be explicitly included. **VaatApi.h** has definitions exclusive to the API whereas **SAVparams.h** has definitions common to both the API and lower-level functions that are not exposed by the API. Static or dynamic linking can be used with this code. During static linking, **vaatApi.lib** is a stub library that is scanned for linkage information to the **vaatAPI.dll** DLL. OpenCV 2.4.9 libraries must also be scanned. At the developer's option, dynamic linking to **vaatApi.dll** may be used instead. Dynamic linking does not require use of **vaatApi.lib** although the function prototypes in that file are a useful reference for the dynamic linking option.

For Linux, **libvaat.so.x.y.z** is a shared object library currently distributed with the file name libvaat.so.1.0.1. A typical installation scenario is to create symbolic link libvaat.so to this file (linker name) in a library directory such as /usr/local/lib. The command ldconfig(8) can then be executed against this file to create libvaat.so.1 and to update the system's shared object cache. The most current pre-built version of OpenCV for Linux at the time of this writing is 2.4.8 and that is the version against which applications have been tested.

C# VAAT application source code can use static methods and structs in **vaatApi.cs**, which implements the Vapi class, for inter-operability to functions in **vaatApi.dll**. Most functions in the Vapi class provide parameter marshaling from C# data types to C++ data types. Many, but not all, definitions in vaatApi.h and SAVparams.h have equivalents in vaatApi.cs and users may extend vaatApi.cs if necessary. Since there is no uniquely correct way to structure these definitions, users may choose to alter or enhance them according to their own project‘s policy and/or experience with these techniques.

For either Windows development methodology **vaatApi.dll** must be in the host's DLL search path at program execution time.

The sample Visual Studio projects included in this distribution reference C:\OpenCV as the root for OpenCV files. OpenCV installers may install to a different root directory by default.

A makefile for building the sample C++ program in Linux is also included.

1.3 Document Organization

The primary audience for this document is Windows software developers although the document is also a reference for Linux software developers. Explanations that use Windows conventions, such Windows file path names, will only include a Linux equivalent when that equivaalent would not be self-evident to the reader. Code examples that use Windows-specific types such as `__int64` or Windows-specific functions are mapped to Linux compatibility by conditional compilation and especially the `win2nix.h` header file.

2 API Description

2.1 General API Characteristics

In a VAAT application, processing of a video stream is associated with a **HANDLE** returned from the **_vaatInitialize** function. Most other API functions use the handle as their first parameter. Any number of video streams may be processed by an application since state variables are segregated by handle value.
Example:

```
HANDLE hdl = _vaatInitialize(argv, &msg);
```

Each VAAT application, or VAAT thread within an application, will normally maintain its own trend database. Initialization and maintenance of a database must be explicitly specified by the application at startup. A database may be specified by an entry in a configuration file (section 2.3) or by a startup parameter (section 2.4.2). Thereafter the database is maintained by the VAAT library in a way that is opaque to the application. Refer to section 3 for detailed discussion of the trend database.

A VAAT application that does not use a database does not support anomaly detection.

2.2 Run-time Support

A platform executing a VAAT application requires that the following DLLs be in one of the directories listed in the system's PATH environment variable (or in the application's startup directory).

- msvcp100.dll
- msvcr100.dll
- opencv_core249.dll
- opencv_ffmpeg249.dll
- opencv_highgui249.dll
- opencv_imgproc249.dll

2.3 Configuration Files

2.3.1 General Description

VAAT application initialization can be controlled by the use of configuration files and/or command-line parameters..

A configuration file consists of keyword/parameters pairs in an ASCII text file. One pair, with keyword and value separated by blanks and/or tabs, comprises one line in the file. Lines that are blank or start with the '#' character are comment lines and are ignored. A comment may be added to a line containing a keyword/parameter pair if separated from the parameter value by at least one blank or tab and if starting with the '#' character. Any line that does not contain a valid keyword/parameter pair and is not a comment line or blank will lead to an error when the configuration file is read.

Keywords are alpha-numeric and cannot have embedded white space. Values can be one of three types:

1 Integer – An integer consisting of a string of characters in the range 0-9 and possible leading minus sign, or in octal format consisting of 0 followed by a string of characters in the range 0-7, or in hexadecimal format consisting of 0x or 0X followed by a string of characters in the range 0-F.
2 Double – A float or double precision number distinguished by use of a decimal point someplace within a string of characters in the range of 0-9 with a possible leading minus sign.
3 String – Any sequence of printable characters, except double quote, enclosed in double quotes.

Example configuration lines are:
```
TREND_DIR    “C:\vaatData\TrendDB”
EDGE_MINTHRESH_WEBCAM    20  # noise threshold may differ among cameras
```

2.3.2 Configurable Parameters

Note: Configurable parameters are mostly not implemented in this release.

The configurable parameters that have default values are listed below with default value and comment:

```
DISPLAY_LEVEL_DFLT 4      #SAV scale level that is displayed
EDGE_MINTHRESH_IPCAM 25   #noise threshold may be different for cameras
EDGE_MINTHRESH_WEBCAM 20  #noise threshold may be different for cameras
SAVBAR_PERIOD_SEC 1       #period in secs of SAVbar feature averaging
TRENDPRL_PERIOD_SEC 60    #period in seconds of trend feature averaging
TREND_NAVG_INIT 3         #no.frames to average for initial trend calc.
TREND_FAST_TCONST 0.8     #fast time constant for initial trend update
TREND_SLOW_TCONST 0.99    #slow time const, steady-state trend update
```

The configurable parameters that do not have default values are:

```
SAVAVG_DIR <string-value> #path and initial file prefix for SAV logging
TREND_DIR  <string-value> #path for trend database
VIDEO      <string-value> #video input specification
```

There is one special configurable parameter that can only be used as a command line parameter:

```
CONFIG=<alternate-config-file>
```

The use of the “config” keyword is fully explained in section 2.4.2.

2.3.3 Usage Examples

The use of many API functions described in this document is illustrated in the sample C++ and sample C# applications included with this distribution, see section 4 for details. As illustrated in the C# example, pointers are passed among functions as type **IntPtr** in contexts where the pointer value and object pointed to are opaque.

2.4 Initialization Functions

2.4.1 extractArgs

```
Arg_t extractArgs(int *argc, char *argv[], const char *argSpec)
```

Extract arguments per argSpec from arguments organized as in a C/C++ argc, argv[] run-time argument set.

- argc – a pointer to the size of the argv array. The value referenced may be modified by the function.
- argv – typically an array of program run-time argument strings. Conventionally, the 0th member is the program name. This array may be shortened by the function.
- argSpec – a string containing argument key letters specifying the arguments that should be removed from argv. The string is in the style of the Linux getopt() function in which a single letter specifies an option letter that does not have an argument value and a single letter followed by ':' specifies an option letter that has an argument value.

The return value type is defined in SAVparams.h. The return value consists of argc and argv[] members containing extracted parameters. The purpose of this function is to pre-process an argc, argv parameter set to remove parameters that require different processing. The argc, argv set used as parameters to the function may be processed differently than the argc, argv set returned by the function.

2.4.2 _vaatInitialize

```
HANDLE _vaatInitialize(char *argArray[], int *badArgIndex);
```

Initialize the API and video stream attributes according to keyword/parameter pairs specified in argArray and the configuration file.

- argArray – Pointer to an array of character strings containing the process name in element 0, keyword/value parameter pairs in elements 1..<array-size>-1, and NULL in element <array-size>.
- badArgIndex – If one or more run-time arguments provided on the command line are invalid, badArgIndex is set to the index in argvArray of the first invalid parameter.

argArrray contents are the same format as those set by the operating system for the main() function of a C/C++ command-line program. For a C# program, argArray can be constructed using the MarshallStrArray function described in section 2.4.4.

The **video=<value>** keyword/value parameter pair is of particular interest when executing this function. Value may be one of:

- A single digit 0-9 which identifies an attached webcam.
- A string <login>[:<password>][@<cam-id>]@<ipAddress>[:portNum] which identifies an IP camera. This form is recognized by the presence of the '@'

character immediately followed by a plausible IP4 address. The default for portNum is 80. Axis™ is the default camera type but a different camera type may be specified by the optional substring @<cam-id>, see comments below.
- Neither of the above is interpreted as a video file name.

An IP camera type other than Axis may be specified by the optional camera ID preceded by @. <cam-id> may be one of:

- **A** or **a** to specify Axis (default).
- **D** or **d** to specify D-Link.
- **X** or **x** to specify an alternate Axis initialization string.
- A string that specifies the exact MJPG CGI path.

Any specifier that starts with an upper-case character results in display of the full MJPG path on the console. For example, "video.cgi" is a generic MJPG path components for Vivotek cameras so the parameter

```
video=admin@Video.mjpg@192.168.0.20
```

may correctly initialize a Vivotek camera. Capitalization of the first character will result in display of the following.

```
IPcam path: http://admin@192.168.0.20/Video.cgi
```

A configuration file, if found, is processed first. vaatInit() searches for configuration file **C:\users\<user-name>\.vaat.ini** (**${HOME}/.vaat.ini** in Linux) unless the **config=<file-name>** parameter is given on the command line as an alternate configuration file. After configuration file processing, parameters given on the command line, which must use keyword=value style, are gathered in order as if they constituted a second configuration file and they are processed. Thus, any keyword appearing both in a configuration file and on the command line is set to the value given on the command line.

2.4.3 _vaatInitializeEx

```
HANDLE _vaatInitializeEx(char *argArray[], int *badArgIndex);
```

Initialize the API and video stream attributes according to keyword/parameter pairs specified in argArray and the configuration file. This function's behavior is the same as described in the previous section for **vaatInitialize()** with one exception. The values in argArray[1] .. argArray[max] need not all be keyword/parameter pairs that are recognized by the VAAT API. Parameters in other formats may be used and these parameters are not processed by the API. When the function returns, argArray is rearranged such that all parameters processed by the API are removed and remaining parameters are at the lower indexes of the array. The last element of the array that points to a parameter is followed by an element containing NULL. An application may then process the argArray for application specific options in the same way that a C/C++ char *argv[] array is processed.

Members of argArray that the API does not attempt to process have one of the following syntax characteristics:

- The argument begins with a - character.
- The argument does not contain an = character.

C# programs can use the function MarshallStrArrayRev() described in section 2.4.5 to map the rearranged argArray into an array of strings.

2.4.4 MarshallStrArray

```
public static IntPtr MarshalStrArray(bool addPname, string[] args);
```

This function is used only in C# applications. Its purpose is to marshall an array of strings in native .NET format into the format conventionally used in C/C++ programs, especially in the context of argc/argv startup parameters.

- addPname – Set the first element of the the array to the program name if true. This parameter should be true when converting to argc/argv format.
- args – A .NET array of strings such as the args array that is passed to a C# Main function.

An application's startup parameters are processed by the vaatInitialize function described in section 2.4.2. Since vaatInitialize is written in C++, parameters passed to it are expected to be conventional C/C++ char[] arrays with null string terminator. MarshallStrArray is a convenience function that provides the needed translation for C# or other .NET languages.

2.4.5 MarshallStrArrayRev

```
public static string[] MarshalStrArrayRev(IntPtr args);
```

This function is used only in C# applications. The function maps a C/C++ char *argv[] array in unmanaged memory to a C# array of strings.

- args – A pointer to unmanaged memory containing a C/C++ array of character pointers logically terminated by an element containing NULL.

If args has a null value or if it points to a location in unmanaged memory that contains a NULL value, the function returns null. An intended use of this function is to recover application-specific start-up parameters that **_vaatInitializeEx()** (section 2.4.3) does not process.

2.4.6 _vaatInitSavStore

```
const char *_vaatInitSAVstore(HANDLE, char *prefix, int period, int
retention, __int64 maxSize);
```

Initialize the SAVbar storage subsystem:

- prefix – the initial substring used for file names. The remainder of the name is constructed from a time stamp, e.g. for **prefix** *SAVbarLog*, a typical file name is *SAVbarLog16Apr2014_14_00_00* for a file with initial data recorded at 2PM on April 16, 2014.
- period – the duration for each file in minutes, 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, or 60, i.e. a number of minutes that divides evenly into 60. Alternatively, the value 0 specifies a single file for the calendar day.
- retention – the number of files to be retained as a history. Files created more than

period file creations ago are automatically deleted. A 0 disables auto deletion.
- maxSize – a maximum size in bytes imposed for each file. If a file exceeds maxSize bytes, a new file is created regardless of other criteria. A 0 specifies that file size is unlimited.

The SAVbar storage subsystem records SAVs at the SAVbar frequency (default 1 second) in disk files. The SAVs are serialized and recorded in a packed format to conserve disk space. The options provided by this function allow an application developer to manage naming, sizing, and retention time for these files. Note that creation of these files is optional in a VAAT application since the files are not needed for real-time anomaly detection. The files could be used by an application that analyzes or uses SAVs in a way not directly supported by the VAAT library. For example, the the sample program **ag.py** (see section 5) graphs trend data and real-time activity level for visual comparison. Ag.py extracts current activity metrics from SAV files as those files are populated.

The function returns NULL if successful. If an error occurred, it returns a pointer to a descriptive error string.

2.5 Configuration Functions

VAAT algorithms can be configued or parameterized by configuration files as explained in section 2.3. Applications retrieve configured parameter values with the functions described in this section.

2.5.1 _vaatGetConfigDbl

```
double _vaatGetConfigDbl(HANDLE, const char *configName);
```

Returns a configured parameter with a double (floating point) type.
- configName – the name of the configurable parameter for which a double value is to be retrieved.

Returns DBL_MIN (the minimum double value) if configName is not defined as a configurable parameter or it is defined but it is not of type double.

2.5.2 _vaatGetConfigInt

```
int _vaatGetConfigInt(HANDLE, const char *configName);
```

Returns a configured parameter with an integer type.
- configName – the name of the configurable parameter for which an integer value is to be retrieved.

Returns INT_MIN (the minimum integer value) if configName is not defined as a configurable parameter or it is defined but it is not of type int.

2.5.3 _vaatGetConfigStr

```
const char *_vaatGetConfigStr(HANDLE, const char *configName);
```

Returns a configured parameter with a C/C++ string type.

- configName – the name of the configurable parameter for which a string value is to be retrieved.

Returns NULL if configName is not defined as a configurable parameter or it is defined but it is not of type C/C++ string.

2.6 Processing Functions

2.6.1 _vaatSAVbar

```
SAVlevel_t *_vaatSAVbar(HANDLE, int *nBlkNot0);
```

Update the averaging period’s average SAV with a video frame:

- frame – the current video frame as a pointer to an OpenCV IplImage.
- nBlkNot0 - updated by the function to the number of non-zero feature blocks comprising the average SAV. A value of -1 indicates averaging is in progress (i.e., has not completed the SAVbar averaging time period specified) and the return value of the function should be ignored – the return value is NULL in this case. A non-negative value indicates that the return value is valid. A value of 0 implies no dynamic video activity.

The function’s return value is NULL until the end of the averaging period. A decision to use the return value may be based on it being non-null or **nBlkNot0** being non-negative. After a valid return value is used by the application and/or logged, **_vaatClearSAV()** with **LEVELTYPE_BAR** must be called to reset the SAVbar values to zero for the next SAVbar averaging period.

2.6.2 _vaatTrendPrlUpdate

int _vaatTrendPrlUpdate(HANDLE);

Update the trend averaging period's SAV with the next video frame. The function's return value is false until the end of the trend averaging period. The function updates the trend database when it returns true. After returning true, **_vaatClearSAV()** with **LEVELTYPE_TREND_PRL** must be called to reset the trend values to zero for the next trend averaging period.

2.6.3 _vaatStoreSAVbar

```
void _vaatStoreSAVbar(HANDLE);
```

Writes the most recently computed SAV to a file if **_vaatInitSAVstore()** was previously invoked. This function should be called after **_vaatSAVbar()** returns a valid SAV and before **_vaatClearSAV()** (see below) is called.

2.6.4 _vaatClearSAV

```
void _vaatClearSAV(HANDLE, int SAVtype);
```

Clears accumulated values for the SAV calculations.

- SAVtype – predefined value **LEVELTYPE_BAR** to clear average calculation for the averaging period or **LEVELTYPE_TREND_PRL** to clear trend calculation for the trend monitoring period.

Must be called with **LEVELTYPE_BAR** after **_vaatSAVbar()** returns a non-null value or **nBlkNot0** is updated to a non-negtive value. Must be called with LEVELTYPE_TREND_PRL after **_vaatTrendPrlUpdate()** returns a true value.

2.6.5 _vaatGetPredominantFeature

```
int _vaatGetPredominantFeature(HANDLE, int featsLevel, int featNum, int
SAVtype, int densThresh);
```

Returns the maximum value in the current SAV at a specified level and for a specified feature.

- featsLevel – the SAV hierarchy level to be examined, e.g. 0 to levels-1.
- featNum – an integer identifying the feature parameter to be evaluated; 1=density, 2=direction, 3=velocity, 4=color, 5-8=corresponding deviations.
- SAVtype – predefined value **LEVELTYPE_BAR** to specify currently computed average or **LEVELTYPE_TREND_PRL** to specify currently computed trend.
- densThresh – the minimum density value required in a image block to trigger an evaluation by this function.

Return values for density, velocity and all deviations are 0-32767. Return values for direction are 0-8 where 0 is no direction and 1-8 are compass directions E, SE, S, etc. (clockwise from E). Return values for color are 0-3 per color with RGB occupying the 3rd least, 2nd least, and least significant pair of bits respectively. If the block has a color that is predominantly skin tone, the returned value is predefined value SKIN_COLOR.

2.6.6 _vaatListLevel

```
int _vaatListLevel(SAVlevel_t *sav, int featsLevel, struct FeatsList
*featsArray, int featsArrayLen, int densThresh);
```

Examines an SAV hierarchy at a specified level and populates an array of FeatsList structs for each image block that has non-zero density.

- sav – a pointer to the SAV object hierarchy examined.
- featsLevel – the hierarchy level of image blocks, 0 .. lastLevel, that is examined.
- featsArray – an array of FeatsList structures that the function populates.
- featsArrayLen – the number of elements in featsArray.
- densThresh – the minimum density value required in a feature block to trigger an evaluation by this function.

2.7 Termination Functions

2.7.1 _vaatTerminate

```
void _vaatTerminate(HANDLE);
```

Frees all dynamic SAV library resources.

2.8 Helper Functions

2.8.1 _vaatGetVideoParams

```
VidParam_t *_vaatGetVideoParams(HANDLE);
```

Returns a pointer to a VidParam_t object for the video device initialized by a call to **_vaatInitialize()**. The VidParam_t structure is documented in vaatApi.h

2.8.2 _vaatDisplay

```
void _vaatDisplay(HANDLE, int wantPrivacy, short *isInitialized);
```

Displays the current video, direction, velocity, color, and fast plot graphical windows on the system console.

- wantPrivacy – set to 0 to display current video or 1 to enable the privacy filter. In privacy mode the current video window displays a static background that is overlaid with motion edges indicating the location of activity.
- isInitialized – should be set to 0 before the first call of this function. Updated by the function to indicate whether initialization occurred.

2.8.3 _vaatDisplayHelp

```
void _vaatDisplay(int argc, char *argv[], char *helpStrings[]);
```

Inspects run-time parameters and displays help messages stored in **helpStrings[]**.

- argc – as passed to a C/C++ main() function.
- argv – as passed to a C/C++ main() function.
- helpStrings – an array of C strings to be displayed as multiple lines of a help message.

The function examines the first run-time parameter, if given, for various forms of a request for help, e.g. **/h**, **-h, –help**, etc. are recognized. If help is requested the function displays each element in the **helpStrings[]** array on a line of console output.

A second form of help displays configurable parameters. To display these parameters add the string **params** after the help specifier separated by a space. Each configurable parameter name will be displayed with its default value and if a configuration file is found in **`C:\users\<user-name>\.vaat.ini`** (or **`${HOME}/.vaat.ini`** in Linux), the over-ride values from the configuration file will also be displayed.

2.8.4 _vaatCheckBgReset

```
void _vaatCheckBgReset(HANDLE, VidParam_t *vidParams, short
*manualBgReset);
```

Enables manual background reset, which resets motion feature calculation. Should be called in the frame acquisition loop.

- vidParams – A pointer to the VidParam_t struct returned by **_vaatGetVideoParams()**.
- manualBgReset – Initially set to 1 to indicate that a background reset is desired.

2.8.5 _vaatGetFrameRate

```
int _vaatGetFrameRate(VidParam_t *vidParams);
```

Returns the frame rate of the video source. The value is always 0 for camera-sources video otherwise it is the frame rate provided from video file metadata.

- vidParams – A pointer to the VidParam_t struct returned by **_vaatGetVideoParams()**.

2.8.6 _vaatGetVideoIn

```
CvCapture *_vaatGetVideoIn(HANDLE);
```

Returns a pointer to an OpenCV struct needed as the argument to other OpenCV functions, e.g. **cvQueryFrame** used by applications for frame acquisition.

2.8.7 _vaatGetCurrentVideoFrame

```
IplImage *_vaatGetCurrentVideoFrame(HANDLE);
```

Returns a pointer to the OpenCV struct that is the abstraction of the current video frame. This pointer is used as an argument by other functions in the API.

2.8.8 _vaatGetHbitMap

```
HBITMAP _vaatGetHbitMap(HANDLE, int mapType);
```

Returns a Windows HBITMAP for one of the five video windows that are displayed by _vaatDisplay() described in section 2.8.2.

- mapType – One of five video map types defined in vaatApi.h and vaatApi.cs.

The bitmaps are updated during per-frame SAV processing performed by **_vaaSAVbar()** and **_vaatTrendPrlUpdate()** described in sections 2.6.1 and 2.6.2.

2.8.9 _vaatGetCurrentVideoFrameAddr

```
IplImage **_vaatGetCurrentVideoFrameAddr(HANDLE);
```

Returns a pointer to the pointer to the OpenCV struct that is the abstraction of the current video frame. This pointer is needed when the value of the pointer to the current video frame must be modified. This function should not be used by managed code.

2.8.10 _vaatStoreThruCurrentVideoFrameAddr

```
void _vaatStoreThruCurrentVideoFrameAddr(HANDLE, IplImage *image);
```

Modifies the value of the pointer to the current video frame to the value of the argument. This function supports pointer-less programing in managed code although it can also be used in an unmanaged C++ context.

2.9 File Management Functions

The functions descrbed in this section are provided for post-processing applications that work with files of SAVs created by the video acquisition application.

2.9.1 _vaatGetHandle

```
const HANDLE _vaatGetHandle();
```

Returns a limited-capability handle for using the remaining file management functions described in this section.

2.9.2 _vaatSAVfileOpen

```
int _vaatSAVfileOpen(HANDLE, char *pathName);
```

- pathName – the name of file of SAV averages.

Returns 0 if successful else the function returns an error code. See additional comments in next section.

2.9.3 _vaatSAVfilePrepare

```
void _vaatSAVfilePrepare(HANDLE, FILE *fp);
```

- fp – a file pointer opened in binary mode for a file of SAV averages.

_vaatSAVfileOpen() uses a path name and **_vaatSAVfilePrepare()** uses an open file descriptor to initialize the library for reading and unpacking packed SAV records that are recorded by **_vaatStoreSAVbar()**, see section 2.6.3.

2.9.4 _vaatGetSAV

```
SAVlevel_t *_vaatGetSAV(HANDLE, time_t *ts, int *nLevels, int *length,
int *nFrames);
```

Read a packed SAV from a SAVbar file and unpack it.

- ts – set by the function to the value of the stored time stamp which is the number of seconds in the computer epoch, i.e. number of seconds since 1 Jan 1970.
- nLevels – if not NULL set to the number of levels in the packed, stored SAVbar SAV record.
- length – if not NULL, set to the length of the packed, stored SAVbar SAV record.
- nFrames – if not NULL, set to the number of video frames used to compute the SAVbar SAV

Returns a pointer to the unpacked SAV.

2.9.5 _vaatRescaleSAV

```
void _vaatRescaleSAV(SAVlevel_t *sav, int nLevels);
```

SAVs are strored in files in a compressed format, see section 2.9.8. This function scales SAV data to its original format which has a precision of 16 bits for all values.

- sav - pointer to a compressed SAV retrieved from file.
- nLevels – depth of SAV in levels.

2.9.6 _vaatGetPredominantFeatureBySAV

```
int _vaatGetPredominantFeatureBySAV(int SAVlevel, int featNum, SAVlevel
*level, int densThresh);
```

This function provides the same capabilty as **_vaatGetPredominantFeature** (see section 2.6.5) except that the SAV structure specified by parameter **level** is evaluated rather than a current SAV structure.

2.9.7 Example

The following C++ program demonstrates use of the file management functions.

```
/*
  The name of a SAVbar file is given as a command-line parameter.  The
  program reads the file and displays the time-stamp, density at level
  0, number of SAV levels, packed SAV length, and number of video
  frames in each SAVbar SAV.  This example has no error processing.
 */
#include <windows.h>
#include <stdio.h>
#define NO_OPENCV
#include "vaatApi.h"

int
main(int argc, char *argv[]) {
    time_t ts;
    int nLevels, len, nFrames;
    SAVlevel *level;
    if(argc != 2) return 1;
    HANDLE sm = _vaatGetHandle();
    if(sm == NULL) return 2;
    int failure = _vaatSAVfileOpen(sm, argv[1]);
    if(failure) return 3;
    while(level = _vaatGetSAV(sm, &ts, &nLevels, &len, &nFrame)) {
        _vaatRescaleSAV(level, nLevels);
        int density = level[0]->block[0].blockFeats.density;
        printf("%I64d %d %d %d\n", ts, density, nLevels, len, nFrames);
    }
    return 0;
}
```

2.9.8 Data Representation

Data in SAVbar files is scaled for efficient storage in disk file. The following table identifies the data stored and its precision.

Parameter	Precision
Motion Density	8 bits
Velocity	4 bits
Direction	4 bits
Color	8 bits
Density Confidence	4 bits
Velocity Confidence	4 bits
Direction Confidence	4 bits
Color Confidence	4 bits

Table 2 SAVbar File SAV Parameters

_vaatRescaleSAV (section 2.9.5) restores values to full precision. Full precision implies a 16-bit normalized value although values are never negative so the maximum value is 2^{15}-1.

2.10 Advanced File Management Functions

The functions in this section provide the capability to create SAVs and pack them so they can be written to files. Most applications are not expected to need these functions since SAV files are normally produced concurrent with video processing.

2.10.1 _vaatCreateSAV

```
int _vaatCreateSAV(SAVlevel **level, int widSS, int htSS);
```

Create an empty SAV structure object.

- level – pointer to SAVlevel pointer. The underlying pointer is modified to point to the dynamically allocated SAV.
- widSS – the sub-sampled width of frames. May be computed by an integer divide of the frame width by the sub-sampling factor (see section 2.10.7).
- htSS – the sub-sampled height of frames. May be computed by an integer divide of the frame height by the sub-sampling factor (see section 2.10.7).

The function returns the number of levels (depth) in the SAV object. All feature values in the newly allocated object are set to 0.

2.10.2 _vaatZeroSAV

```
void _vaatCreateSAV(SAVlevel *level, int nLevels);
```

Set feature values, for nLevels of depth, to 0 in a SAV object.

- level – pointer to SAV object..
- nLevels – the number of levels to descend for setting feature values to 0.

2.10.3 _vaatSerializeSAV

```
SAVpackRslt_t *_vaatSerializeSAV(HANDLE, int time, int nLevels,
SAVlevel *level, int packType);
```

Map a SAV structure to an array of bytes so that it can be written to a file.

- time – a 32-bit time stamp. Conventionally, the number of seconds since 1/1/1970 is used.
- nLevels – the number of SAV levels to serialize.
- level – a pointer to the SAV structure.
- packType – specifies packing style (see section 2.10.4).

The function returns a pointer to a SAVpackRslt_t type which is defined in SAVparams.h. The members of this data type provide the length of the array of bytes and its address. A negative length value is an error code.

2.10.4 _vaatGetPackerStyle

```
int _vaatGetPackerStyle(int isWideData, int isPruned);
```

Returns a SAV packing style based on whether data should be scaled to a space saving format and whether "pruning" should be used to further conserve space.

- isWideData – set to 1 if no scaling is desired, 0 for scaling.
- isPruned – set to 1 if pruning is desired, 0 for no pruning.

Pruning is a technique whereby feature values for lower level blocks are not stored if the density of the "parent" block is 0. Pruning should only be used if the following condition is met:

- Number of levels = L
- Horizontal and vertical resolution evenly divisible by $5x2^L$.

2.10.5 _vaatGetSAVfileAttr

```
SAVfileAttr_t *_vaatGetSAVfileAttr(HANDLE);
```

Retrieves attributes of how SAVs are stored in a file opened by _vaatSAVfileOpen (section 2.9.2) or _vaatSAVfilePrepare(section 2.9.3). The function returns a ponter to a SAVfileAttr_t object which is defined in SAVparams.h. The structure members are sub-sampled image width, sub-sampled image height, isWideData value, and isPruned value. The latter two values are defined in section 2.10.4. Note that sub-sampled dimensions are 1/5th of pixel dimensions.

2.10.6 _ vaatGetRawSAVavgFileHeader

```
const unsigned char *_vaatGetRawSAVavgFileHeader(HANDLE,  const
SAVfileAttr_t *savFileAttr);
```

Returns an array of unsigned characters that is the linear representation of a SAVfileAttr_t object.

- `savFileAttr` – populated SAVfileAttr_t object specifying SAV geometry and packing style.

The linear form of this data is used as the header for a SAV file. Linearized SAVs may be written in time order following the header.

The SAVfileAttr_t type is defined in SAVparams.h. The length of this data is defined in vaatApi.h as SAVAVG_FILE_HEADER_LEN.

2.10.7 _vaatGetSSfactor

```
const unsigned char *_vaatGetRawSAVavgFileHeader()
```

Returns the sub-sampling factor used for SAV computations. The original video frame resolution is this factor multiplied by the sub-sampled dimensions such as those in a SAVfileAttr_t object (see section 2.10.5). The dimensions that are a result of this multiplication may have been rounded down to be an even multiple of the sub-sampling factor.

3 Trend Database

A practical VAAT application requires a trend database. A separate trend directory must exist for each invocation of **_vaatInitialize()** that specifies use of a trend database. This directory should initially be empty except for the optional file **.trendBucketMap** explained in section 3.2. If an application is terminated and then restarted it will normally use its existing database.

3.1 Database Organization

A trend database entry consists mainly of a serialized SAV stored in an individual file. The file also contains a small amount of other state information and it contains a preamble of metadata that is used internally and may be used for TBD audit operations. A database entry contains trend information for one minute (60 seconds).
A full trend database can consist of 10080 entries, i.e. 7 days x 24 hrs/day x 60 entries/hr although it can consist of fewer entries as explained in section 3.2. The database is constructed as a tree hierarchy of directories and files (entries) as follows,

- Seven directories, one for each day named **day_N**, 0≤N≤6 for Sunday..Saturday.
- 24 directories for each day named **hr_N**, 0≤N≤23 for each hour of the day.
- 60 files (entries) for each hour named **min_N**, 0≤N≤59 for each minute of the hour.

Basic backup of a trend database can be performed by recursive file copy.

3.2 Consolidation of Trend Information

It may be desirable to build a database consisting of fewer than 10080 entries when scenes are monitored that have similar activity for certain blocks of time. For example, you may want to build a database that categorizes days as either weekday or weekend. For a given day you may want fine detail during working hours, perhaps 0800-1900 but you may want to treat all night-time hours equivalently.

If it exists, the API interprets a file named **`.trendBucketMap`** in the top level of the trend database directory. The file consists of 7 data lines each with 24 fields. The file may also have blank lines or non-blank lines starting with the '#' which are not interpreted.

The 7 data lines specify trend remapping for days 0-6 in the trend database. Day 0 normally corresponds to Sunday. A remapping specification has the form:
<target-day>:<target-hour>

For example, assume the 2nd data line in the file is:
0:0 0:0 0:0 0:0 0:0 0:0 0:0 0:0 0:8 0:8 0:8 0:8 0:8 0:1 0:1 0:1 …
The 2nd line specifies trend remapping for Monday.

The first 8 specifications are all 0:0 which express that these hours are expected to trend to about the same activity and the contribution to trend for all should be consolidated in the database location for day0, hour 0. In this scenario day 0 is assumed to represent weekdays and hour 0 is assumed to represent early morning.

The next 5 specifications similarly express that mid-morning activities are all expected to have about the same activity and the contribution to trend for those hours should be consolidated in the database location for day0, hour8. Subsequent trend for the afternoon is specified for consolidation as day0, hour1.

4 Sample Applications

A sample C++ and sample C# program are included with the VAAT distribution. The programs have equivalent functionality. Depending on startup options, they can generate SAVbar files and they can create and manage a trend database.

4.1 Program Distribution

Sample program SAVbase is supplied with source code in a Visual Studio 2010 solution named SAVbase.

Sample program vaatDemoCsharp is supplied with source code in a Visual Studio 2010 solution named vaatDemoCsharp.

Both solutions have the VAAT library, i.e. vaatApi.lib and vaatApi.dll for SAVbase, and vaatApi.dll for vaatDemoCsharp pre-installed in their Release and Debug directories.

A simple makefile is provided for building SAVbase in Linux. Due to idiosyncrasies in MSVC++ pre-processor design, the statement `#include “StdAfx.h”` is visible in SAVbase.cpp during Linux compilation. The statement should either be commented or removed for Linux or an empty StdAfx.h file should be provided.

4.2 Startup

An example Windows startup command for the C++ application program is,

```
>SAVbase savavg_dir=\vaatData\camera1\savbar\sav  trend_dir=\vaatData\camera1\trend
```

The directory path **\vaatData\camera1** must exist and it must contain subdirectories **savbar** and **trend**. This example assumes a webcam is connected and is the desired video input device. A different video input may be selected with the **video=…** startup option.

4.3 Execution

The SAVbase sample program displays 5 windows in addition to the console window:

- Current Video – This window displays current video from the input webcam, IPcam, or video file. Overlaid upon the video are the motion edges, shown in two colors. The blue indicates motion edges of the current frame. The white is “motion blur” edges, and is a collection of motion edges from some small number of previous frames (about 10). Motion blur is used for calculation of 1st and 2nd order features, direction and velocity.

- Direction, Velocity, and Color – These windows display the SAV features that are calculated at the SAVbar frequency (1 second default).
  1. The Direction map shows motion direction in compass directions in image space as follows: red (east), orange (SE), yellow (S), chartreuse (SW), green (W), cyan (NW), blue (N), purple (NE).

2. The velocity map shows magnitude of velocity by gradation in color from purple (slow) to red (fast), with the same color sequence between these as above for direction.
3. The color map indicates colors of motion pixels (that is, not of background). The colors are the same as those for direction and velocity, with the addition of white, gray, and pink (skin). Black is not a color; it is the absence of motion.

- FastPlot – This window displays the SAVbar density feature value plotted against time, with the current value scrolling from the right to the left of the plot, and with default sample frequency 1 per second. The duration of the plot is 200 frames, or about 6 seconds at 30 frames per second. The pink bins show the current density value at level 0. The blue horizontal line shows the value of the trend. The default frequency of trend value update is once per minute.

- Console – In the console window, press ‘q’ or ‘x’ to terminate execution.

4.4 Video File Source

If the VAAT application is initialized with a video file as the video source, creation of SAVbar files and trend accumulation may also be specified. In this case trend updates start at day_0, hr_0, min_0 and triggering of updates is based on frame counting rather than the system’s real-time clock.

5 Activity Graph

This distribution includes a python program named **ag.py** for display of current activity and trend data. The program has a real-time mode of operation and a report mode. Prerequisites for using the program are an installation of Python version 2.7.x and the Python Imaging Library (PIL).

5.1 *Operation*

The activity graph can be started in real-time mode from a Windows command window as follows:

```
>python ag.py <savbar-dir\file-prefix> <trend-db-dir>
```

The first parameter identifies the directory location and file name prefix of the SAVbar files. The second parameter identifies the directory location of the trend database. The program generates a .PNG graphics file named **ActivityGraph.png** and updates it about once a minute as it follows additions of SAVbar information to the SAVbar files. The current graph can be displayed at any time by a graphics file display program. Double clicking on the file icon in Windows invokes the default program for the .PNG file type to display the graph.

The activity graph can be started in report mode from a Windows command window as follows:

```
>python ag.py <savbar-dir\file-prefix> <trend-db-dir> <mo>/<day>
```

The first two parameters are the same as for real-time mode. The third parameter specifies a month and day for which a graphics file named **ActivityGraph_DDMonYYYY.png** should be generated.

5.2 *Algorithm*

At startup the program reads the trend database and draws trend traces for both the previous and current days. It then reads the SAVbar file(s) for the previous day and draws that day's current activity trace. It then reads the SAVbar file(s) for the current day and draws that day's current activity trace up to the current time. Thereafter, the program "sleeps" and "awakens" once a minute to update the current activity trace. At midnight, the graph for the current day is finalized and then used as the graph for the previous day for the next 24 hours and a new graph for the current day is started.

The values plotted are derived from SAV level 1 density. There are 9 blocks at SAV level 1 and the program picks the maximum intensity (total activity) value from 9 blocks for both trend and current activity. These values are graphed at the resolution of the graphics file meaning that some fine resolution is not shown.

Values plotted for trend are retrieved from the trend database subject to the remapping considerations described in section 3.2.