ОБЩАЯ ИНФОРМАЦИЯ О СИСТЕМЕ

Система видеоаналитики BDLAB VAS (сокр. VAS) предназначена для предоставления услуг по интеллектуальной обработке видеопотоков через каналы Интернет, а также на закрытых сетях операторов и провайдеров связи.

К сервисной аналитике VAS относятся модули, оценивающие качество получаемого с камеры изображения, и сообщающие о каких-либо нежелательных отклонения в получаемом изображении. В нее входят следующие модули:

- Детектор расфокусировки камеры
- Детектор засветки объектива камеры
- Детектор загрязнения объектива камеры
- Детектор отворота камеры
- Детектор заслонения камеры

Детектор расфокусировки камеры сравнивает последовательные кадры видеопотока и сообщает о расфокусировке в момент, когда она произошла. Когда камера снова сфокусируется придет соотвествующее сообщение.

Детектор засветки объектива камеры сообщает координаты областей с засветами. Засветы оцениваются по яркости пикселей, пороговое значение настраивается.

Детектор загрязнения объектива камеры камеры сообщает координаты областей с загрязнениями. Загрязнениями считаются фоновые темные пиксели, которые появились во время работы модуля. Пороговое значение настраивается.

Детектор отворота камеры при помощи сравнения фона на последовательных изображениях отправляет сообщения, что камера отвернута. Возврат (или не возврат) в исходное положение не детектируется.

Детектор заслонения камеры отправляет сообщение в случае резкого сильного заслонения камеры. Такой эффект. Модуль также может срабатывать в случае резкого сильного загрязнения объектива.

ПРИНЦИП РАБОТЫ СИСТЕМЫ

Система принимает http запрос, содержаний ссылку на видеопоток и код модуля аналитики, которым требуется обработать данный видеопоток. После этого система раз в заданный промежуток времени отсылает информацию о статусе задачи и о требуемых событиях, когда они происходят.

ТРЕБОВАНИЯ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ, НЕОБХОДИМОМУ ДЛЯ УСТАНОВКИ СИСТЕМЫ

VAS функционирует и выполняет возложенные на него функции на базе серверов под управлением операционной системы Linux Ubuntu 16.04 или новее.

ТРЕБОВАНИЯ К ВИДЕОПОТОКАМ

VAS ожидает видеопотоки, сжатые в соответствии со стандартном H. 264, содержащие в своем составе специальную SEI метку в заголовке каждого NAL-юнита. Каждая метка представляет собой точный timestamp кадра в милисекундах.

РАБОТА С API СИСТЕМЫ

VAS принимает http запрос, обрабатывает кадры видеопотока методами компьютерного зрения, фиксирует наступление искомых событий, преобразовывает сообщения о них в формат JSON и отправляет их в виде сообщений брокеру системы AMQ. Задача аналитики может иметь как конечное время жизни (если в его рамках обрабатывается конечный фрагмент видеоконтента из архива, ограниченный задаваемым в запросе временным

интервалом), так и неограниченное время жизни (если в его рамках обрабатывается видеоконтент, непрерывно транслируемый с реального источника видеоконтента в режиме realtime). В последнем случае канал заканчивает работу по команде от клиента.

Допустимые HTTP-методы запроса:
GET – получить информацию о сущности, не изменяя ее
POST – добавить/изменить/удалить сущность и возможно получить по ней информацию

Допустимые HTTP-коды ответа:
200 OK – передается всегда, когда HTTP-обмен данным оказался успешным (обработка запроса и отправка ответа на уровне HTTP). Если при выполнении команды на уровне VAS возникла ошибка, код ответа также должен быть 200 OK, тело ответа должно содержать информацию об ошибке в формате JSON:

```
{
    id: <идентификатор задания>,
    success: false,
    error: <краткая информация об ошибке (код)>,
    errorDescription: <подробная информация об ошибке, опционально>,
}
```

любой другой HTTP-код ответа – должен передаваться только в случае соответствующей ошибки на уровне HTTP.

Коды ошибок

0 ecUnhandledException
1 ecNotFound
2 ecVideoUnreachable
3 ecInvalidVideoFormat
4 ecIncorrectVideoUrl
5 ecNoFreeTaskSlots

Описание сущностей и методов

1 Создание задания

путь: */task/{id}*

метод: *POST*

параметры в пути запроса:

* String *id – UUID-идентификатор задания*

параметры в теле запроса (JSON):

* Integer *vendor* – Код вендора аналитики,
* Integer *metadataType* – Код типа аналитики,
* String *cameraId* – UUID-идентификатор камеры из архива которой должен быть загружен фрагмент обрабатываемого видеоконтента (далее архивный режим) либо с которой должен транслироваться в потоке обрабатываемый видеоконтент (далее потоковый режим),

CameraUrl[] *cameraUrls* – Массив URL, для доступа к видеоконтенту камеры. Если параметр присутствует, сервис должен использовать один из элементов для получения видеоконтента. В случае недоступности видеоконтента по первому URL, необходимо использовать следующий URL из массива и так далее. При отсутствии параметра cameraUrl, формирование URL для доступа к видеоконтенту осуществляется по cameraId с помощью дополнительных средств (CCTV API Service).
CameraUrl – структура, содержащая необходимые ссылки для доступа и управления выдачей видеоконтента. Включает в себя:

* String *video* – Ссылка на видеопоток,
String *control* – Ссылка для управления видеопотоком.

String *taskKey* – Ключ, идентифицирующий задание, если не хватает UUID-идентификатора создаваемого задания. Фактически описывает множество аналитической информации, полученной в результате работы данного канала с данной камерой. Может включать в себя например последовательность taskId-subTaskId-interval и тп,
Long *startTime* – Unix-timestamp момента начала фрагмента обрабатываемого видеоконтента в архивном режиме,
Long *endTime* – Unix-timestamp момента конца фрагмента обрабатываемого видеоконтента в архивном режиме. Присутствие параметра endTime является для VAS указанием на работу в архивном режиме. При отсутствии endTime, startTime игнорируется,
* Map<String, Object> *rules* – Правила мониторинга на стороне вендора. Набор пар ключ значение, где ключ – уникальный UUID-идентификатор правила, значение – само правило, имеющее специфичный формат для каждого типа аналитики.
Object *customParams* – Дополнительная информация.

ответ:

в случае успешного выполнения:

* String *id* – Идентификатор задания,
* Integer *state* – Текущее состояние задания,
* Boolean *success* – true,
* Long *start* – Unix-timestamp момента начала работы задания.

в случае сбоя: информация об ошибке, формат см. п. Обработка ошибок

1.3 Удаление (остановка) задания

путь: */task/{id}*

метод: *DELETE*

параметры в пути запроса:

* String *id – UUID-идентификатор задания*

ответ:

в случае успешного выполнения:

* String *id* – Идентификатор задания,
* Integer *state* – Текущее состояние задания,
* Boolean *success* – true,
* Long *start* – Unix-timestamp момента начала работы задания.
Long *framesProcessed* – Количество успешно обработанных кадров,
Float *progress* – Доля в процентах обработанной части от общего объема обрабатываемого фрагмента видеоконтента, для потокового режима всегда 0,
Long *tsLastFrame* – Unix-timestamp момента последнего обработанного кадра.

в случае сбоя: информация об ошибке, формат см. п. Обработка ошибок

1.4 Получение информации о состоянии задания

путь: */task/{id}*

метод: *GET*

параметры в пути запроса:

*** String *id – UUID-идентификатор задания*

ответ:

в случае успешного выполнения:

* String *id* – Идентификатор задания,
* Integer *state* – Текущее состояние задания,
* Boolean *success* – true,
* Long *start* – Unix-timestamp момента начала работы задания.
Long *framesProcessed* – Количество успешно обработанных кадров,
Float *progress* – Доля в процентах обработанной части от общего объема обрабатываемого фрагмента видеконтента, для потокового режима всегда 0,
Long *tsLastFrame* – Unix-timestamp момента последнего обработанного кадра,
* String *host* – Hostname/ip сервера, на котором запущен процесс VAS,
* Integer *port* – Порт, на котором процесс VAS слушает запросы API,
* Long *tsCreated* – Unix-timestamp момента создания данного ответа/сообщения,
* Integer *vendor* – Код вендора аналитики,
* String *cameraId* – UUID-идентификатор камеры,
String *taskKey* – Ключ, идентифицирующий задание.

1.5 Обновление ссылок на видеопотоки

путь: */task/{id}/cameraUrls*

метод: *PUT*

параметры в пути запроса:

*** String *id* – UUID-идентификатор задания

параметры в теле запроса (JSON):

* CameraUrl[] *cameraUrls* – Массив URL, для доступа к видеоконтенту камеры. Если параметр присутствует, сервис должен использовать один из элементов для получения видеоконтента. В случае недоступности видеоконтента по первому URL, необходимо использовать следующий URL из массива и так далее. При отсутствии параметра cameraUrl, формирование URL для доступа к видеоконтенту осуществляется по cameraId с помощью дополнительных средств (CCTV API Service).
CameraUrl – структура, содержащая необходимые ссылки для доступа и управления выдачей видеоконтента. Включает в себя:
    * String *video* – Ссылка на видеопоток,
    String *control* – Ссылка для управления видеопотоком.

ответ:

в случае успешного выполнения:

* String *id* – Идентификатор задания,
* Integer *state* – Текущее состояние задания,

* Boolean *success* – true,
* Long *start* – Unix-timestamp момента начала работы задания.

Выдача результатов модуля.

По мере обработки видеопотока модуль отправляют данные фрагментами. Фрагмент может содержать любое целое число событий (событие – описание состояния данного объекта в данном кадре). Событие однозначно идентифицируется ключом из двух полей: таймстампом кадра и идентификатором объекта. Данный ключ должен быть уникальным, то есть из одного кадра не должно быть сгенерировано больше одного события с данным идентификатором объекта.
Фрагмент дополняется заголовком, преобразуется в поток байт (кодировка текста UTF-8), представляющий собой готовое сообщение для системы обмена сообщениями. Сообщение отправляется в систему обмена.
Данные доступа к системе обмена сообщениями (хост, порт, логин, пароль) и имя приемного обменника (exchange)/темы (topic) должны настраиваться в конфигурации VAS.
Система обмена сообщениями: rabbitmq.
Формат заголовка сообщения с метаинформацией:

Поле	Тип	Длина	Описание
messageType	unsigned char	1 байт	тип сообщения, 0 для сообщения с событиями, сгруппированными по кадрам, 1 для сообщения с событиями, сгруппированными по объектам
marker	unsigned int	4 байта	значение константа 0xFFFFA5FF
headerVersion	unsigned char	1 байт	значение константа 2
cameraUrl-len	unsigned short	2 байта	длина поля cameraUrl
cameraUrl	unsigned char[]	cameraUlr-len	URL доступа к видеопотоку, который был использован
vendorId	unsigned char	1 байт	код вендора аналитики
taskType	unsigned char	1 байт	тип задания: 0 для realtime, 1 для archive
cameraId	unsigned char[]	36 байт	UUID, идентификатор камеры
taskKey-len	unsigned short	2 байта	длина поля taskKey
taskKey	unsigned char[]	taskKey-len	ключ задания, передается при создании.
format	unsigned char	1 байт	значение константа 1 (JSON)

1) выходной формат для детектора расфокусировки

Детектор имеет два варианта, один – основанный на Лапласиане, второй - на анализе частот на изображении. Сообщения отправляются только при пекреключении одного из флагов lapl или fred из значение True в значение False или наоборот.

```
{
"time": - Целое число, время первого кадра с событиями расфокусировки или фокусировки (в милисекундах),
"lapl" : True/False, - True отправляется в случае расфокусировки, False в случаи фокусировки.
"freq" : True/False, - True отправляется в случае расфокусировки, False в
}
```

2) выходной формат для детектора засвета объектива

Детектор возвращает координаты прямоугольников с засветами.

```
{
"time": - Целое число, время первого кадра с событиями засвета (в милисекундах),
"over_coord": - массив координат прямоугольников, в которых встретились засветы
}
```

3) выходной формат для детектора загрязнения объектива

Детектор возвращает координаты прямоугольников с загрязнениями.

```
{
"time": - Целое число, время первого кадра с событиями загрязнения (в милисекундах),
```

```
"pol_coord": - массив координат прямоугольников, в которых встретились
загрязнения
}
```

4) **выходной формат для детектора отворота камеры**

Сообщение приходит разово, в случае отворота камеры, и параметр motion может иметь только значение True.

```
{
"time": - Целое число, время первого кадра события отворота(в
милисекундах),
"motion": - True.
}
```

5) **выходной формат для детектора заслонения камеры**

Сообщение приходит при возникновении заслонения или при изчезновении ранее обнаруженного заслонения

```
{
"time": - Целое число, время первого кадра заслонения или время первого
кадра изчезновения ранее детектированного заслонения
"obstruction": - True/False. True – заслонение появилось. False –
заслонение исчезло.
}
```

Выдача специальной информации

Состояние сервиса

VAS с настраиваемой периодичностью (по умолчанию 5 секунд) отправляет сообщения о своем состоянии в систему AMQ. Имя приемного обменника настраивается отдельно (далее это ap.vas.state). Поля сообщения:

* Float *cpuUsage* – Средняя загрузка CPU всех ядер сервера 0...1,
* Float *memoryUtilization* – Доля использованной оперативной физической памяти сервера 0...1,
* Integer *numberOfTasks* – Число работающих заданий,
* String *runner* – Host:port сокет, на котором VAS слушает запросы по API,
* Long *tsCreated* – Unix-timestamp, момент создания данного сообщения

Состояние задания

VAS с настраиваемой периодичностью (по умолчанию 5 секунд) отправляет сообщения о состоянии каждого своего работающего в данный момент задания в систему обмена сообщениями (только rabbitMQ). Имя приемного обменника настраивается отдельно (далее это ap.vas.task.state). Поля сообщения:

* String *id* – Идентификатор задания,
* Integer *state* – Текущее состояние задания,
* Boolean *success* – true,
* Long *start* – Unix-timestamp момента начала работы задания.

Long *framesProcessed* – Количество успешно обработанных кадров,
Float *progress* – Доля в процентах обработанной части от общего объема обрабатываемого фрагмента видеконтента, для потокового режима всегда 0,
Long *tsLastFrame* – Unix-timestamp момента последнего обработанного кадра,

* String *host* – Hostname/ip сервера, на котором запущен процесс VAS,
* Integer *port* – Порт, на котором процесс VAS слушает запросы API,
* Long *tsCreated* – Unix-timestamp момента создания данного ответа/сообщения,
* Integer *vendor* – Код вендора аналитики,
* String *cameraId* – UUID-идентификатор камеры,

String *taskKey* – Ключ, идентифицирующий задание.

VAS должен отправлять сообщения о состоянии задания все время от его создания до перехода в конечное состояние.
Если VAS прекращает выполнение задания в результате неустранимой ошибки и переводит данное задание в конечное состояние error, он должен отправить сообщение о состоянии задания, содержащее описанные выше поля. При этом значение поля success должно быть false. В этом случае дополнительно отправляются поля:

* String *error* – краткая информация об ошибке (код),

String *errorDescription* – подробная информация об ошибке, опционально.

Сигнал завершения обработки задания
VAS отправляет сообщения о завершении обработки задания незамедлительно после завершения. в систему AMQ, обменник ap.metadata. В json сообщении поле type будет равняться custom_message_end_of_task.

Установка и запуск системы

Системные требования:

OC: Ubuntu 20.04
Nvidia drivers >= 460.73.01
Docker version >= 20.10.6

Установка и запуск:

1) Загружаем образ по ссылке http://62.173.140.127/vas-runtime-1.1.tar
2) Загружаем сохраненный образ в докер

```
docker load --input vas-runtime-1.1.tar
```

3) Создаем и запускаем контейнер на основе этого образа

Сервис слушает команды на 8080 порту, параметр -p 127.0.0.1:8080:8080 открывает этот порт на локальном хосте, при необходимости можно изменить порт и хост.

```
docker run -p 127.0.0.1:8080:8080 --gpus all --runtime nvidia -e
NVIDIA_DRIVER_CAPABILITIES=video,compute,utility --name vas-runtime vas-runtime-1.1
```

4) Получаем айпи контейнера

```
docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' vas-runtime
```

5) Заходим в контейнер по ssh

```
ssh root@172.17.0.3
password: x
```

6) Переходим в директорию /root/vas_root/data

```
cd /root/vas_root/data/
```

7) Запускаем сервис

```
./vas.sh ../config/srv.lua
```

8) Переходим в каталог /root/vas_root/frontend

```
cd /root/vas_root/frontend
```

переходим в каталог /root/vas_root/frontend, открываем файл Config.json находим поле amqpUri и вписываем туда по образцу настройки своей инсталляции rabbitmq.

```
#Запускаем сервис команд
./frontend.sh
```