Общее руководство пользователя системы «РоадАР Аналитика - выпавший груз»

Содержание

Введение

Система разработана компанией ООО “РоадАР” для распознавания и анализа различных ситуаций на дороге.

Система поддерживает механизмы взаимодействия со сторонними информационными системами. Посредством вызова API поставляемой библиотеки. А также текущая версия системы предполагает выбор и реализацию конкретного механизма взаимодействия с внешними системами по согласованию с заказчиками.

1 Назначение и условия применения системы

1.1 Назначение системы

Обработка изображений для выявления и фиксации факта появления постороннего неподвижного предмета (выпавший/оставленный предмет) на полосе движения и/или наблюдаемой зоне.

1.2 Базовый функционал

- обработка изображений;
- выявление и фиксация факта появления постороннего неподвижного предмета (выпавший/оставленный предмет) на полосе движения и/или наблюдаемой зоне;
- информационный обмен с внешними системами.

1.3 Функциональные характеристики:

Текущая версия системы позволяет решать задачу по распознаванию события “выпавший груз” - появление неподвижного объекта на полосе движения и/или наблюдаемой зоне.

1.4 Программное обеспечение рабочего места

Библиотека может быть встроена в программное обеспечение, установленное на рабочем месте пользователя. В этом случае клиентская часть **системы** может использоваться на любом рабочем месте, имеющем подключение к сети Internet (или сети передачи данных предприятия). Разрешающая способность видеосистемы и монитора – не ниже 1280x1024. Рекомендуется широкоформатный монитор.

Вид ПО	Программный продукт
ОС (приведены варианты)	Windows 10 Linux (Ubuntu, Debian, Альт, ROSA, UBLinux, ICLinux).

Таблица 1. Системные требования

Данные требования могут меняться в зависимости от особенностей программного обеспечения, которое использует библиотеку системы .

1.5 Виды пользователей

Пользователем системы является пользователь программного обеспечения, в которое встраивается библиотека **системы** . Описание доступных возможностей API выполняется для пользователя, обладающего максимально возможными правами по доступу к программе.

1.6 Описание системы

API Системы написано на C++.

~ src/core/modules/analytic_specific_params.hpp:

```
…
namespace an::core {
/*!
  * \brief Общая структура для хранения общих параметров для дорожных аналитик
  */
 struct RoadAnalyticParams {
  RoadAnalyticParams() = default;
  /**
   * \brief Конструктор с параметрами
   * \param [in] grnd указатель на класс для проекции объектов на дорогу
   * \param [in] roadLanes множество полос, поданных на вход аналитике
   * выпавших грузов
   */
  RoadAnalyticParams(std::shared_ptr<GroundCalibration> &grnd,
                     std::vector<RoadLane> lanes)
      : ground(grnd), roadLanes(std::move(lanes)) {
    spdlog::info("Road lanes count for road analytic: {}", roadLanes.size());
  }

  /// Conversion API -> Core
  explicit RoadAnalyticParams(const an::RoadAnalyticParams &params) {
    std::transform(params.roadLanes.begin(),
                   params.roadLanes.end(),
                   std::back_inserter(roadLanes),
                   [](const an::RoadLane &rl) { return RoadLane(rl); });
  }

  bool operator==(const RoadAnalyticParams &other) const {
    bool eqGround = ground == other.ground
```

```
        || (ground && other.ground && *ground == *other.ground);
    return eqGround && roadLanes == other.roadLanes;
  }

  /// указатель для проецирования объектов на дорогу (через калибровочную
  /// матрицу)
  std::shared_ptr<GroundCalibration> ground;
  /// множество полос дороги
  std::vector<RoadLane> roadLanes;
};

/*!
 * \brief Структура для хранения параметров аналитики "Выпавший груз"
 */
struct LostCargoAnalyticParams : RoadAnalyticParams {
  /**
   * \brief Конструктор с параметрами
   * \param [in] roadParams общие параметры для дорожной аналитики
   * \param [in] minCargoSize минимальный размер груза для создания события
   * обнаружения выпавшего груза
   */
  LostCargoAnalyticParams(RoadAnalyticParams roadParams,
                          float minConfidence,
                          float minCargoWidth,
                          float maxCargoWidth,
                          float minCargoHeight,
                          float maxCargoHeight)
      : RoadAnalyticParams(std::move(roadParams)),
        minConfidence(minConfidence),
        minCargoWidth(minCargoWidth),
        maxCargoWidth(maxCargoWidth),
        minCargoHeight(minCargoHeight),
        maxCargoHeight(maxCargoHeight) {}

  /// Conversion API -> Core
  explicit LostCargoAnalyticParams(const an::LostCargoAnalyticParams &params)
      : RoadAnalyticParams(params),
        minConfidence(params.minConfidence),
        minCargoWidth(params.minCargoWidth),
```

```
        maxCargoWidth(params.maxCargoWidth),
        minCargoHeight(params.minCargoHeight),
        maxCargoHeight(params.maxCargoHeight) {}

    bool operator==(const LostCargoAnalyticParams &other) const {
      if (!RoadAnalyticParams::operator==(other))
        return false;

      return minConfidence == other.minConfidence
          && minCargoWidth == other.minCargoWidth
          && maxCargoWidth == other.maxCargoWidth
          && minCargoHeight == other.minCargoHeight
          && maxCargoHeight == other.maxCargoHeight;
    }

    float minConfidence;

    float minCargoWidth;
    float maxCargoWidth;

    float minCargoHeight;
    float maxCargoHeight;
  };

}
```

~ src/core/modules/lost_cargo/cargo_analytic_controller.hpp:

```
#pragma once

#include "common/platform.hpp"

#include "modules/analytic_controller.hpp"

namespace an { namespace core {
  /**
   * \brief Контроллер для работы аналитики выпавшего груза
   */
  class LostCargo : public AnalyticController {
```

```
  public:
        explicit LostCargo(an::core::FrameQueue inputQueue,
                an::core::FrameQueue outputQueue,
                AnalyticEventQueue analyticEventQueue,
                std::shared_ptr<InferenceEngine::Core> &ie,
                std::string xmlPath);

        ~LostCargo() override;

        AnalyticType getControllerAnalyticType() const override;

        void updateGroundCalibration(
        const std::string &streamId,
        const std::shared_ptr<GroundCalibration> &groundCalibration);

  protected:
        std::unique_ptr<Analytic> makeSpecificAnalytic(
        const AnalyticSpecificParams &params) override;

        /// объект инференса для классификатора
        std::shared_ptr<InferenceEngine::Core> ie_;
        /// путь до нейросети классификатора груза
        std::string netPath_;
  };
}}  // namespace an::core
```

~ src/core/modules/lost_cargo/cargo_analytic.hpp

```
#pragma once

#include "cargo_detector.hpp"
#include "ground/ground_projection.hpp"
#include "modules/analytic.hpp"
#include "modules/analytic_specific_params.hpp"
#include "nn/openvino/classifier.hpp"
#include "utils/prob_compute.hpp"

namespace an { namespace core {
```

```
struct CargoCandidate {
	CargoDetect detect;
	cv::Rect2f checkRect;
	cv::Rect2f largeCheckRect;
	cv::Mat grayTemplate;
	int templateUpdateCount = 0;
	std::vector<cv::Mat> historyImages;
	std::vector<cv::Mat> historyLargeImages;

	uint64_t checkFrameNum = 0;
	int successCheck = 0;
	int failCheck = 0;

	ProbCompute probCalc;
	bool forceGenerate = false;

	std::string uuid;
};

struct CargoHistoryMat {
	cv::Mat img;
	double time;
	DetectedObjects detections;
};

class LostCargoAnalytic : public Analytic {
 public:
	LostCargoAnalytic(std::string analyticID,
		AnalyticEventQueue eventQueue,
		bool needSavePreview,
		const LostCargoAnalyticParams &params,
		std::shared_ptr<InferenceEngine::Core> &engine,
		std::string &xmlPath);

	void processFrame(const an::core::pFrame &) override;
	void updateGroundCalibration(
	const std::shared_ptr<an::core::GroundCalibration> &);

	void finishAllCargo();
```

```
        cv::Mat getLastBackgroundChange();

private:
        std::mutex mutex_;
        CargoDetector detector_;
        std::vector<RoadLane> lanes_;
        std::vector<DetectedObjects> detectHistory_;
        std::vector<CargoCandidate> watchlist_;
        std::vector<CargoCandidate> checklist_;
        std::unordered_map<std::string, std::shared_ptr<AnalyticEvent>>
        activeEvents_;
        std::deque<CargoHistoryMat> historyMats_;
        std::shared_ptr<GroundCalibration> ground_;
        std::shared_ptr<Classifier> classifier_;
        cv::Mat lastGrayscale_;
        double lastHistTime_{-1};
        double saveHistEverySec_{10};  // sec
        uint64_t saveDetectsNextFrameId_{};
        uint64_t lastProcessedFrameId_{};
        uint64_t processEveryNFrame_{9};  // skip this amount of frame
        std::atomic_bool requestEndAllCargo_;
        std::vector<CargoDetect> lastFinishedDetects_;
        cv::Rect manualGenerateCandidate_{};
        bool lanesNormalized_{};
        bool isEventGenerationEnabled_{true};

        float filterConf_{};

        float minCargoWidth_;
        float maxCargoWidth_;

        float minCargoHeight_;
        float maxCargoHeight_;

        bool canGenerateStartEvent(const pFrame &,
                                   const CargoCandidate &,
                                   float * /*netConfidence*/ = nullptr);

        std::shared_ptr<AnalyticEvent> generateStartEvent(const pFrame &,
```

```
                                    const CargoCandidate &,
                                    float confidence);

    std::shared_ptr<AnalyticEvent> finishEvent(const pFrame &,
                                const CargoCandidate &);

    static float matchTemplateCrop(const cv::Mat &grayImg,
                    cv::Rect crop,
                    const cv::Mat &templateMat);

    AnalyticEvents processDetects(const std::vector<CargoDetect> &,
                    const pFrame &);

    void updateWatchlist(const std::vector<CargoDetect> &, const pFrame &);

    AnalyticEvents generateStartEvents(const pFrame &);

    AnalyticEvents generateEndEvents(const pFrame &);

    float getConfidence(const CargoDetect &);

    bool checkLastFinishedDetects(const CargoDetect &);

    void debugDraw(const pFrame &, const std::vector<CargoDetect> & = {});

    cv::Mat getGrayscaleFrame(cv::Rect2f = {});

    /// сохраняем в истории нашу картинку
    void saveCurrentFrameHistory(const pFrame &);

    /// достаем из истории участок изображения
    /// для которого наименьшее пересечение с детектором обьектов
    std::vector<cv::Mat> getFromHistory(
    const cv::Rect2f &, const std::vector<cv::Rect2f> & /*cropRects*/);

    cv::Mat getCurrentProcessed(cv::Rect2f);

    void drawLanes(cv::Mat & /*frame*/);
};
```

}} // namespace an::core

2 Инструкция по установке системы

2.1. Установка Docker

В инструкции подразумевается, что пользователь использует ОС Linux, Ubuntu 20.04 (для других дистрибутивов, инструкции могут отличаться)

Обязательным предусловием для установки серверной платформы MDT является установка следующих пакетов:

- ***docker 18.06.1-ce+***;

На Ubuntu их можно установить следующими командами:

```
$ sudo apt-get update
$ sudo apt-get install \
        apt-transport-https \
        ca-certificates \
        curl \
        gnupg-agent \
        software-properties-common
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
$ sudo add-apt-repository \
        "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
        $(lsb_release -cs) \
        stable"
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Детали для установки Докера и добавления вашего пользователя в группу Докер можно найти по ссылкам:

1) https://docs.docker.com/compose/install

2) https://docs.docker.com/install/linux/docker-ce/ubuntu.

2.2 Скачивание и сбор частей архив, загрузка образа и его запуск

1) Скачайте из папки 11 ZIP-архивов.
https://www.roadar.info/file-share/6af005a9-0083-4929-863a-07b628e2cc84

Для загрузки файлов необходимо авторизоваться. Логин и пароль передаются при покупке системы.

Загруженные файлы представляют собой части архива с компонентами системы. Пароль от собранного архива так же передаётся при покупке системы.

2) Необходимо установить архиватор. В терминале Linux Ubuntu это команда:

sudo apt install unzip

3) С помощью терминала зайдите в папку с 11 ZIP-архивами:
lost_cargo.tar.parta a
lost_cargo.tar.parta b
lost_cargo.tar.parta c
lost_cargo.tar.parta d
…
lost_cargo.tar.parta k

4) Собираем полный архив из частей:

cat lost_cargo.tar.parta*[1] > lost_cargo.tar

5) Затем загружаем полученный докер-образ в список образов докера

docker load -i lost_cargo.tar

6) Далее запускаем образ с параметром требуемой аналитики :

docker run analytics “*lost_cargo”*

[1] * - подразумевает букву в конце наименования одного из 11 архивов (от “a” до “k”)

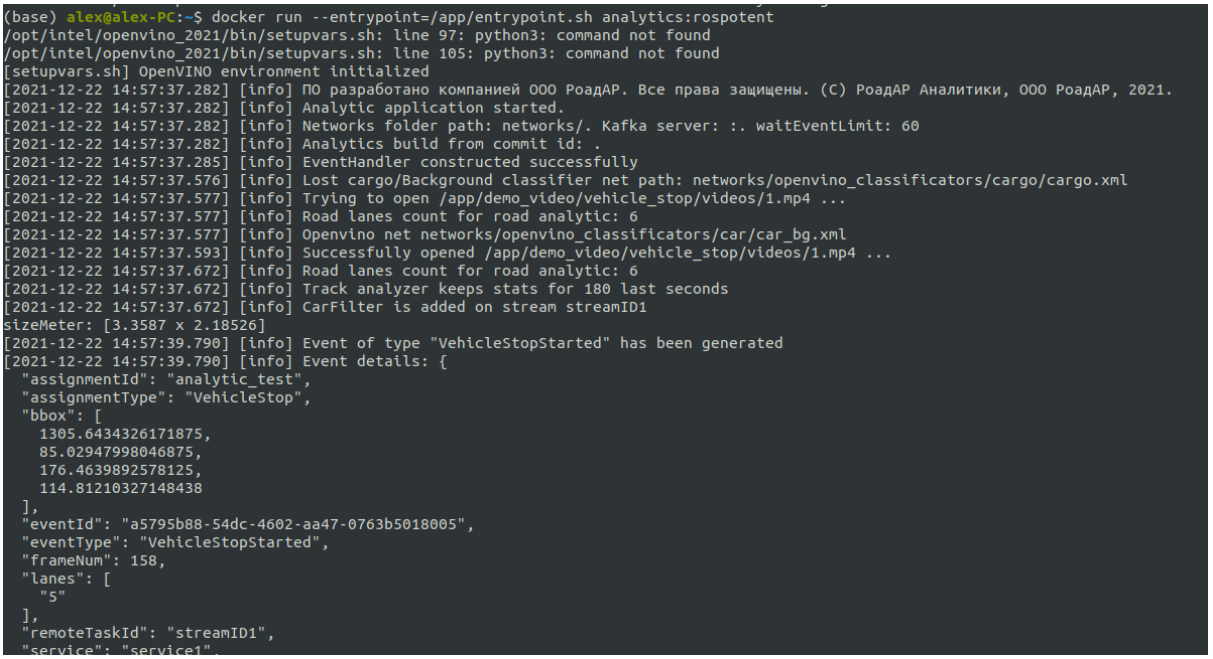

Рисунок 2. Пример вывода в консоль удачного запуска (дождитесь вывода статуса).

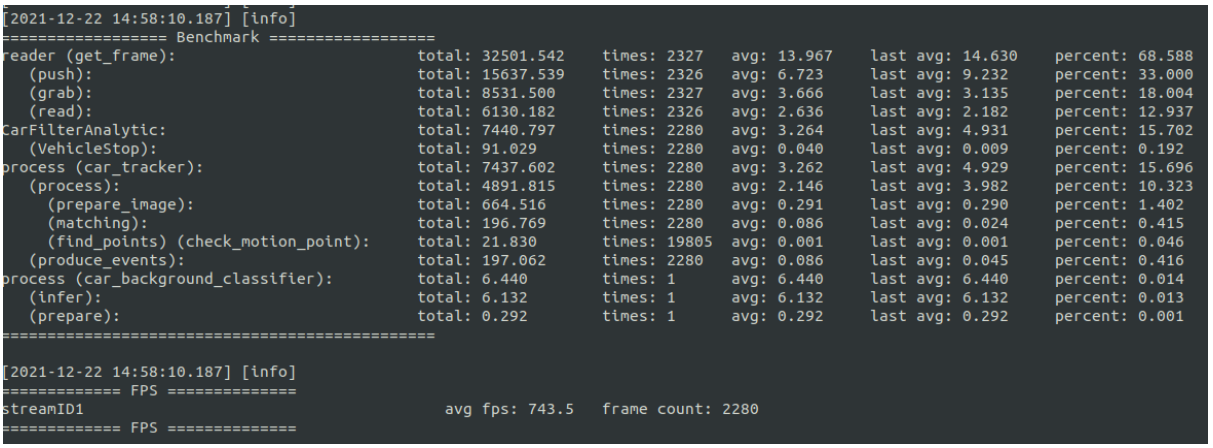

Рисунок 3. Пример вывода текущего статуса (обновляется каждые 2 минуты).

3 Контактная информация производителя программного продукта

3.1 Юридическая информация

- **Название компании:** ООО «РоадАР».
- **ИНН** 1615013172
- **ОГРН** 1161690183665
- **Юр. адрес:** 420500, г Иннополис, ул Университетская, д 7, офис 332

3.2 Контактная информация службы технической поддержки

- **Сайт:** roadar.info
- **Email:** info@roadar.info
- **Тел.:** +7-903-307-16-75

Фактический адрес размещения инфраструктуры разработки:
420500, г. Иннополис, ул. Университетская, дом 7, офис 645

Фактический адрес размещения разработчиков:
420500, г. Иннополис, ул. Университетская, дом 7, офис 645

Фактический адрес размещения службы поддержки:
420500, г. Иннополис, ул. Университетская, дом 7, офис 645