**Общее руководство пользователя системы «РоадАР Аналитика - проезды ТС»**

**Содержание**

[**Введение 3**](#_je8m6wcagh1x)

[**1 Назначение и условия применения системы “РоадАР Аналитика - падение средней скорости транспортного потока” 4**](#_3dy6vkm)

[1.1 Назначение системы 4](#_1t3h5sf)

[1.2 Базовый функционал 4](#_4d34og8)

[1.3 Функциональные характеристики 4](#_czhijeca4yrq)

[1.4 Программное обеспечение рабочего места 4](#_2s8eyo1)

[1.5 Виды пользователей 5](#_17dp8vu)

[1.6 Описание системы 5](#_3rdcrjn)

[**2 Инструкция по установке системы 8**](#_8hnbzf3klg5a)

[2.1 Установка Docker 8](#_b98xe46k97cc)

[2.2 Скачивание и сбор частей архив, загрузка образа и его запуск 9](#_z83rduyufh6j)

[**3 Контактная информация производителя программного продукта 11**](#_ecomzkltjoke)

[3.1 Юридическая информация 11](#_3rdcrjn)

[3.2 Контактная информация службы технической поддержки 11](#_2tow4ctptd8)

#

# Введение

**“РоадАР Аналитика - проезды ТС”** – система, разработанная компанией ООО “РоадАР” для распознавания и фиксации номерного знака, марки и скорости транспортного средства (далее - ТС), проехавшего в наблюдаемой зоне.

**Система “РоадАР Аналитика - проезды ТС”** 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 “РоадАР Аналитика - проезды ТС”

## 1.1 Назначение системы

Обработка изображений для определения и фиксации номерного знака, марки и скорости ТС, проехавшего в наблюдаемой зоне.

## 1.2 Базовый функционал

* обработка изображений;
* определение и фиксация номерного знака, марки и скорости ТС, проехавшего в наблюдаемой зоне;
* информационный обмен с внешними системами.

## 1.3 Функциональные характеристики

Текущая версия системыпозволяет решать задачи по фиксации номерного знака, марки и скорости ТС, проехавшего в 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 “РоадАР Аналитика - проезды ТС”** может использоваться на любом рабочем месте, имеющем подключение к сети Internet (или сети передачи данных предприятия). Разрешающая способность видеосистемы и монитора – не ниже 1280х1024. 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 **“РоадАР Аналитика - проезды ТС”**.

## 1.5 Виды пользователей

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

## 1.6 Описание системы

API Системы написано на C++ и предоставляет несколько основных классов для распознавания номеров.

***~ src/core/modules/analytic\_specific\_params.hpp:***

…

namespace an::core {

/\*!

 \* \brief Общая структура для хранения оющих парметров 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

 \* выпаших 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;

 };

struct VehicleTransitAnalyticParams : RoadAnalyticParams {

 enum class UsedDetector { ANALYTICS, NUMBERS };

 /\*\*

 \* \brief Конструктор с параметрами

 \* \param [in] roadParams общие параметры для дорожной аналитики

 \*/

 explicit VehicleTransitAnalyticParams(RoadAnalyticParams roadParams,

 UsedDetector usedDetector)

 : RoadAnalyticParams(std::move(roadParams)),

 usedDetector(usedDetector) {}

 explicit VehicleTransitAnalyticParams(

 const an::VehicleTransitAnalyticParams &params)

 : RoadAnalyticParams(params),

 usedDetector(convertDetectorType(params.usedDetector)) {}

 static UsedDetector convertDetectorType(

 an::VehicleTransitAnalyticParams::DetectorToUse usedDetector) {

 return usedDetector

 == an::VehicleTransitAnalyticParams::DetectorToUse::NUMBERS

 ? UsedDetector::NUMBERS

 : UsedDetector::ANALYTICS;

 }

 bool operator==(const VehicleTransitAnalyticParams &other) const {

 return RoadAnalyticParams::operator==(other)

 && other.usedDetector == usedDetector;

 }

 UsedDetector usedDetector;

 };

}

***~ src/core/modules/vehicle\_transit/vehicle\_transit\_analytic.hpp:***

#pragma once

#include <deque>

#include <map>

#include <memory>

#include "modules/analytic.hpp"

#include "modules/analytic\_specific\_params.hpp"

#ifdef USE\_ROADAR\_NUMBERS

#include "roadar\_plates\_container.hpp"

#endif

namespace an { namespace core {

 // common functions

 class VehicleTransitAnalytic : public Analytic {

 public:

 explicit VehicleTransitAnalytic(

 std::string analyticID,

 AnalyticEventQueue analyticEventQueue,

#ifdef USE\_ROADAR\_NUMBERS

 const std::shared\_ptr<RoadarPlatesContainer> &platesApi,

#endif

 VehicleTransitAnalyticParams params);

 ~VehicleTransitAnalytic() override;

 void processFrame(const pFrame &frame) override;

 // you can call this function manually to see the calculated things from

 // this analytic

 void debugDraw(cv::Mat &debugFrame, const pFrame &frame) const override;

 void updateGroundCalibration(

 const std::shared\_ptr<an::core::GroundCalibration> &groundCalibration);

#ifdef USE\_ROADAR\_NUMBERS

 void debugDrawNumbers(cv::Mat &debugFrame, const pFrame &frame) const;

#endif

 private:

 bool useNumbersDetector\_;

 // Only with RoadarNumbers

#ifdef USE\_ROADAR\_NUMBERS

 struct TransitFrame {

 boost::optional<RoadarNumbers::Vehicle> vehicle;

 boost::optional<RoadarNumbers::PlateResult> plate;

 boost::optional<RoadarNumbers::VehicleInfoWithProb> info;

 boost::optional<RoadarNumbers::TrackResult> trackResult;

 boost::optional<std::string> color;

 boost::optional<pFrame> frame;

 boost::optional<double> currentSpeed;

 };

 struct FinishedPlate {

 RoadarNumbers::TrackResult track;

 RoadarNumbers::PlateResult selectedPlate;

 boost::optional<pFrame> selectedFrame;

 };

 void processPlateTracksFinished(

 const std::vector<RoadarNumbers::TrackResult> &plateTracksFinished);

 void processFinishedCarTracks();

 /\*\*

 \* check over finished plate tracks, forcefully raise those that do not have

 \* an ongoing car track matched

 \* @param vehicleTracksCurr

 \*/

 void checkOverFinishedPlateTracks(

 const std::vector<RoadarNumbers::VehicleTrack> &vehicleTracksCurr,

 uint64\_t frameNum);

 TransitFrame getBestTransitInfo(const RoadarNumbers::VehicleTrack &track);

 void raiseEvent(const TransitFrame &transit);

 boost::optional<RoadarNumbers::VehicleInfoWithProb> recognizeVehicle(

 const pFrame &frame,

 const std::vector<RoadarNumbers::RNPoint> &plate,

 const RoadarNumbers::TrackResult &trackResult);

 boost::optional<RoadarNumbers::VehicleInfoWithProb> recognizeVehicle(

 const pFrame &frame, const RoadarNumbers::RNRect &rect);

 std::string recognizeColor(const pFrame &frame,

 const std::vector<RoadarNumbers::RNPoint> &plate,

 const RoadarNumbers::TrackResult &trackResult);

 std::string recognizeColor(const pFrame &frame,

 const RoadarNumbers::RNRect &rect);

 bool processTransit(TransitFrame transit);

 void processFrameWithNumbers(const pFrame &frame);

 boost::optional<pFrame> getFrameFromBuffer(uint64\_t frameNum);

 boost::optional<double> frameNumToTimestamp(uint64\_t frameNum);

 boost::optional<uint64\_t> timestampToFrameNum(double ts);

 std::pair<RoadarNumbers::PlateResult, boost::optional<pFrame>>

 getBestFrameFromBuffer(std::vector<RoadarNumbers::PlateResult> plates);

 std::pair<RoadarNumbers::Vehicle, boost::optional<pFrame>>

 getBestFrameFromBuffer(std::vector<RoadarNumbers::Vehicle> vehicles);

 template <typename PlateOrVehicle>

 boost::optional<double> getCurrentSpeed(

 const std::vector<PlateOrVehicle> &track, uint64\_t frameNum);

 std::list<pFrame> frameBuffer\_;

 // time stamps buffer stores each frame number with its time stamp

 std::deque<std::pair<uint64\_t, double>> tsBuffer\_;

 int badTsCounter\_ = 0;

 std::map<int, RoadarNumbers::VehicleTrack> finishedCars\_;

 std::map<int, FinishedPlate> finishedPlates\_;

 std::map<int, uint64\_t> unmatchedPlatesLastChecked\_;

 std::shared\_ptr<RoadarNumbers::IDetector> plateDetector\_;

 std::shared\_ptr<RoadarNumbers::IVehicleDetector> vehicleDetector\_;

 std::shared\_ptr<RoadarNumbers::IVehicleRecognizer> vehicleRecognizer\_;

 bool platesUseGPU\_ = false;

#endif

 /\* Analytic detection algorithm:

 \* 1. Calculate the middle point of the box which you have of a car

 2. Make a vector with this point and the same point kMoveVecFramesCount\_

 frames ago

 3. if this vector length is less than kMinVecLengthForMovingCar\_ then the

 car is not moving actually otherwise it is moving we say

 4. for the last kNumLastMovingStatusFrames\_ frames store if the car is

 moving or not according to this vector length

 5. we go and check this movement history made in step 4

 6. now if kMajorityRatio\_ (out of 100% so 0.7 for 70%) of the these

 recorded status in step 4 where actually moving (let's say majority) then

 that means it went outside the picture because it was moving to leave the

 camera vision

 7. We also remove cars which we didn't detect at all and not updating

 their history since kCarHistoryRemoveLimit frames in order to not make

 memory filled in real application where it runs for days

 \*/

 struct VehicleHistory {

 cv::Rect box;

 bool isMoving;

 float prob;

 uint64\_t frameNum;

 double speed;

 TypeObject objectType;

 CarType carType;

 };

 // functions which only use analytics (kept because even with Numbers using

 // Analytics to detect is still possible)

 /\*\*

 \* Function for drawing lanes, boxes with relevant info in labels to see how

 \* it works

 \* @param debugFrame the frame we wanna draw the debug stuff on (out

 \* parameter)

 \* @param frame the frame from analytics that should the information and

 \* initial picture be taken from

 \*/

 void debugDrawAnalytics(cv::Mat &debugFrame, const pFrame &frame) const;

 void detectVehicleTransitWithAnalytics(const pFrame &frame);

 void raiseEventAnalytics(const pFrame &frame, int64\_t carId);

 /\*\*

 \* Makes a screenshot for the vehicle transit event. First it finds the

 \* earliest frame when the vehicle was moving then it draws a box around it

 \* then saves the screenshot to an out parameter

 \* @param carId the id of the car which raised the event of the transit

 \*/

 void getScreenshotAndInfoForTransitVehicle(int64\_t carId,

 MatCore &screenshotMat,

 VehicleHistory &info);

 // TODO calculate these constants relative to the height of the camera which

 // will improve the algorithm significantly

 // number of frames which the vector of movement will be calculated from

 // (from the end)

 const uint32\_t kMoveVecFramesCount\_ = 5;

 // min length for movement vector to consider the car moving inside our

 // frame

 const float kMinVecLengthForMovingCar\_ = 8.0f;

 // number of the last frames which we should store for if a car was moving

 // or not, earlier than that are discarded

 const uint32\_t kNumLastMovingStatusFrames\_ = 100;

 // number of frames passed without any detection for a car in our frame that

 // we should delete this car history (to avoid memory overflow)

 const uint32\_t kCarHistoryRemoveLimit\_ = kNumLastMovingStatusFrames\_ \* 3;

 // ratio to say whether most of the time the car was moving or not

 const float kMajorityRatio\_ = 0.7f;

 // max number of frames that car can be unseen, if it was unseen more

 // than this number then we check if most of the frames it was moving to

 // know if we should raise the event or not

 const uint32\_t kMaxFramesCarCanBeUnseen\_ = 25;

 // keep pointers of the last kSizeOfFrameBuffers to get a relevant

 // screenshot from when a vehicle that raised the event vehicle transit its

 // size should be kMaxFramesCarCanBeUnseen + kNumLastMovingStatusFrames\_

 std::map<uint64\_t, pFrame> frameNumToFrameBuffer\_;

 std::unordered\_map<int64\_t, uint64\_t> carIdToLastFrameNum\_;

 std::unordered\_map<int64\_t, std::list<VehicleHistory>> idToVehicleHistory\_;

 std::unordered\_set<int64\_t> eventRaisedIds\_;

 // common between numbers and not using numbers

 cv::Size frameSize\_;

 std::mutex mutex\_;

 VehicleTransitAnalyticParams params\_;

 };

}} // namespace an::core

***~ src/core/modules/vehicle\_transit/vehicle\_transit\_analytic\_controller.hpp:***

#pragma once

#include <memory>

#include "modules/analytic\_controller.hpp"

#include "vehicle\_transit\_analytic.hpp"

namespace an { namespace core {

 class VehicleTransitAnalyticController : public AnalyticController {

 public:

 /\*\*

 \* Constructor for Vehicle Transit analytic controller (no need to remove

 \* the related plates params because they default to empty values which will

 \* not be used)

 \* @param inputQueue the queue where the frames are received

 \* @param outputQueue the queue where the frame should be sent after

 \* @param analyticEventQueue the event queue to put the events in

 \* @param usePlates whether plates should be enabled or not (in each

 \* analytic is different but this one is for optimization in case we don't

 \* wanna enable Numbers at all)

 \* @param platesNetworksPath path to networks of plates (in case it is used)

 \* @param cachePath path to where to save cache for networks of plates in

 \* case it is needed

 \* @param platesUseGPU whether to use GPU with plates or not

 \*/

 explicit VehicleTransitAnalyticController(

 FrameQueue inputQueue,

 FrameQueue outputQueue,

 AnalyticEventQueue analyticEventQueue,

 bool usePlates,

 const std::string &platesNetworksPath,

 const std::string &cachePath,

 bool platesUseGPU);

 ~VehicleTransitAnalyticController() override;

 AnalyticType getControllerAnalyticType() const override;

 void updateGroundCalibration(

 const std::string &streamId,

 const std::shared\_ptr<an::core::GroundCalibration> &groundCalibration);

 protected:

 std::unique\_ptr<Analytic> makeSpecificAnalytic(

 const AnalyticSpecificParams &params) override;

 private:

#ifdef USE\_ROADAR\_NUMBERS

 std::shared\_ptr<RoadarPlatesContainer> platesApi\_;

 bool platesUseGPU\_ = false;

 bool usePlates\_ = true;

#else

 bool usePlates\_ = false;

#endif

 };

}} // namespace an::core

# 2 [Инструкция по установке системы](#_26in1rg)

## 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](https://docs.docker.com/install/linux/docker-ce/ubuntu/).

## 2.2 [Скачивание и сбор частей архив, загрузка образа и его запуск](#_26in1rg)

1. Необходимо установить архиватор. В терминале Linux Ubuntu это команда:

*sudo apt install unzip*

1. Скачайте папку Документы РоадАР Аналитика - <https://www.roadar.info/file-share/5ad2fb14-dd07-427d-afb5-a0feb1f99cbe>

Наименование файла: \_\_\_\_\_\_\_\_\_\_

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

Загруженный файл представляет собой архив с компонентами системы. Пароль от него так же передаётся при покупке системы.

 

Рисунок 1. Скачивание папки с демо.

1. С помощью терминала зайдите в папку Документы РоадАР Аналитика - выпавший груз. В текущей рабочей директории вы должны увидеть папку demo\_image\_parts и 11 архивов ZIP:
analytics.tar.parta a

analytics.tar.parta b

analytics.tar.parta c

analytics.tar.parta d

…

analytics.tar.parta k

1. Собираем один архив из его частей:

|  cat analytics.tar.parta\*[[1]](#footnote-0) > analytics.tar |
| --- |

1. Затем загружаем полученный докер-образ в список образов докера

|  docker load -i analytics.tar |
| --- |

1. Далее запускаем образ с параметром требуемой аналитики :

|  docker run analytics “*analytic\_name”*где, “*analytic\_name” -* название аналитики*:*1. ВЫПАВШИЙ ГРУЗ - **lost\_cargo**2. ДТП - **road\_accident**3. ОСТАНОВКА ТС - **vehicle\_stop**4. ПАДЕНИЕ СРЕДНЕЙ СКОРОСТИ ТРАНСПОРТНОГО ПОТОКА - **speed\_decrease**5. ПРОЕЗДЫ ТС - **vehicle\_transit**6. ДВИЖЕНИЕ ТС В ЗАПРЕЩЕННОМ НАПРАВЛЕНИИ - **wrong\_direction**7. ОПРЕДЕЛЕНИЕ ЗАТОРА ТС - **traffic\_jam**8. ИНТЕНСИВНОСТЬ ДВИЖЕНИЯ - **statistics**9. ПРОЕЗДЫ НА ПЕРЕКРЕСТКАХ - **crossroad**Пример: |
| --- |



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 715

Фактический адрес размещения разработчиков:

420500, г. Иннополис, ул. Университетская, дом 7, офис 715

Фактический адрес размещения службы поддержки:

420500, г. Иннополис, ул. Университетская, дом 7, офис 715

1. \* - подразумевает букву в конце наименования одного из 11 архивов (от a до k) [↑](#footnote-ref-0)