"The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 723205"

Deliverable D3.4

Report on risk evaluation system and use cases for pilot test

Due date of deliverable: 31/10/2019

Actual submission date: 29/10/2019

Project details

Project acronym	SAFER-LC
Project full title	SAFER Level Crossing by integrating and optimizing road-rail infrastructure management and design
Grant Agreement no.	723205
Call ID and Topic	H2020-MG-2016-2017, Topic MG-3.4-2016
Project Timeframe	01/05/2017 – 30/04/2020
Duration	36 Months
Coordinator	UIC – Marie-Hélène Bonneau (bonneau@uic.org)

Document details

Title	Report on risk evaluation system and use cases for pilot test
Workpackage	WP3
Date of the document	08/10/2019
Version of the document	03
Responsible partner	UTBM
Status of the document	Final
Dissemination level	Public

Document history

Revision	Date	Description
01	08/10/2019	First Draft
02	28/10/2019	Final Version
03	29/10/2019	Ethical and final review

Consortium - List of partners

Partner No	Short name	Name	Country
1	UIC	International Union of Railways	France
2	VTT	Teknologian tutkimuskeskus VTT Oy	Finland
3	NTNU	Norwegian University of Science and Technology	Norway
4	IFSTTAR	French institute of science and technology for transport, development and networks	France
5	FFE	Fundación Ferrocarriles Españoles	Spain
6	CERTH-HIT	Centre for Research and Technology Hellas - Hellenic Institute of Transport	Greece
7	TRAINOSE	Trainose Transport – Passenger and Freight Transportation Services SA	Greece
8	INTADER	Intermodal Transportation and Logistics Research Association	Turkey
9	CEREMA	Centre for Studies and Expertise on Risks, Environment, Mobility, and Urban and Country planning	France
10	GLS	Geoloc Systems	France
11	RWTH	Rheinisch-Westfaelische Technische Hochschule Aachen University	Germany
12	UNIROMA3	University of Roma Tre	Italy
13	COMM	Commsignia Ltd	Hungary
14	IRU	International Road Transport Union - Projects ASBL	Belgium
15	SNCF	SNCF	France
16	DLR	German Aerospace Center	Germany
17	UTBM	University of Technology of Belfort-Montbéliard	France

Executive summary

This document presents the technical aspects as well as the performance evaluation of the Risk Evaluation System (RES) application developed by UTBM in collaboration with CEREMA Toulouse within task 3.1 of the work package 3.

The main objective of task 3.1 is to extract models of dangerous behaviors from video recordings and construct a database of all detected behaviors for a specific level crossing (LC). Indeed, little knowledge is available about precursors, i.e., behaviors that would be indicative of the dangerousness of a LC since most databases only contain data about accidents that did occur and were investigated as reported in deliverable D1.2 “Level crossing accidents and factors behind them”. For this purpose, RES application is dedicated to detect and categorize observed behaviors by analyzing the space-time trajectories of LC users with respect to the rail, road and LC infrastructures.

Once the system was implemented, several tests were performed using synthetic videos generated through simulation in virtual environments. The system evaluation shows the robustness and the limit cases of the algorithms implemented in the application.

Table of content

1. INTRODUCTION

1.1. Objectives of the SAFER-LC project

Over the past few years, there has been one death and close to one serious injury every day on level crossings in Europe. Therefore, SAFER-LC aims to improve safety and minimize risk by developing a fully integrated cross-modal set of innovative solutions and tools for the proactive management and design of level-crossing infrastructure.

These tools will enable road and rail decision makers to find effective ways to detect potentially dangerous collision situations at level crossings, prevent incidents at level crossings by innovative design and predictive maintenance methods and mitigate the consequences of incidents/disruptions due to accidents or other critical events.

The project will focus both on technical solutions, such as smart detection services and advanced infrastructure-to-vehicle communication systems, and on human practices, to adapt infrastructure design to end-users and to enhance coordination and cooperation between different stakeholders from different transportation modes.

The project will first identify the needs and requirements of rail-road infrastructure managers and LC users and then seek to develop innovative smart detection and communication systems, and to adapt them for use by all types of level crossing users.

A series of pilot tests across Europe will be rolled out to demonstrate how these new technological and non-technological solutions can be integrated, validated for their feasibility and evaluated in terms of their performance.

The project will deliver a bundle of recommended technical specifications (for standardization), human practices and organizational and legal frameworks for implementation.

Finally, SAFER-LC will develop a toolbox accessible through a user-friendly interface which will integrate all the project results and solutions to help both rail and road managers to improve safety at level crossings.

1.2. Description of Task 3.1

Task 3.1 aims at helping rail and road infrastructure managers as well as public authorities to collect data about possible dangerous behaviors occurring at LC. Indeed, most available data related to LC are collected during investigations when an accident happens and are by definition statistically rare. Almost nothing is known about precursors, i.e., behaviors that would be indicative of risks of accidents.

The objective of Task 3.1 is to develop a software named Risk Evaluation System (RES) that analyzes video recordings of activities in the vicinity of level crossings. These videos are expected to be collected by fixedly placed cameras that record on a full 24/7 basis. The output of the RES is a database of all the potentially dangerous situations or driving behaviors contrary to the traffic laws, that are present in the video.

A video presentation of the Task 3.1 is available online[1].

1.3. Input data for the RES

The first challenge we had to overcome in this project is the difficulty to obtain data to test the system. Indeed, in order to record videos one has to obtain authorizations from both the rail and road infrastructure manager. Additionally, one has to comply with the General Data Protection Regulation (EU) 2016/679. This regulation requires either that the data subjects give their consent to the collection of their personal data and are given with full access to either correct or delete them, otherwise all collected data must be anonymized. The first option would introduce a bias in the study rendering the results completely useless, the second would put such a large overhead that a major part of the resources dedicated to Task 3.1. would have to be redirected to this operation. Finally, even in the case that the collection of data on LC would be GDPR compliant, there is no guarantee that hours of recordings would contain a sufficient number of dangerous behaviors to evaluate the RES performances in a statistically rigorous fashion.

We chose instead to generate synthetic video data using an advanced simulator that includes realistic graphics, physically weather and lighting simulation, accurate vehicle dynamics and AI-based driver behaviors. This simulator was developed by UTBM's CIAD laboratory prior to this project and has been adapted for the needs of Task 3.1.

1.4. Interactions with other tasks

Tasks 3.1 and 3.2 are strongly linked, they both attempt to overcome the technical challenge of recognizing dangerous behaviors. Though the end goal of both tasks is quite different, we collaborated with CEREMA in the development of the Smart Detection System (see deliverables D3.1 and D3.5) while they were involved in the development of the RES.

1.5. Structure of the document

This document is structured into two main sections:

Section 2 describes the technical implementation of the RES application,

[1] https://youtu.be/iZCmZvh-AVY

Section 3 presents the results of several experiments performed to validate the components of the system and the system as a whole.

Section 2 is decomposed into five subsections:

- subsection 2.1 gives an overview of the general architecture of the software,
- subsection 2.2 describes how the first stage of the system, the detection stage, is implemented and presents each of the three tasks of this stage: light signal detection, barrier angle detection and object detection,
- subsection 2.3 presents the technical details of the implementation of the tracking stage,
- subsection 2.4 does the same for the final stage of the application: the recognition stage,
- subsection 2.5 presents the graphical user interface of the application and provides information about the type of data a RES user must input and how, using tools that were developed and integrated in the software.

Section 3 is decomposed into four subsections:

- subsection 3.1 describes the process and the algorithms used to generate the synthetic videos that were used to evaluate the system in the subsequent parts of the document,
- subsection 3.2 presents the performance evaluation of the detection stage,
- subsection 3.3 does the same for the tracking stage,
- subsection 3.4 shows the results of the performance evaluation of the recognition stage.

Section 4 concludes the document and outline future works to improve overall performances and add features to the RES application.

2. RISK EVALUATION SYSTEM

2.1. Architecture Overview

The Risk Evaluation System (RES) is an application that analyzes a video recording of a level crossing and its surrounding, and extracts data about the occurrence of dangerous and/or anomalous behaviors. This analysis is performed off-line in a semi-supervised fashion and focuses on general motion, i.e., the analysis operates on space-time trajectories instead of directly analyzing the images to recognize activities. The system builds a database of detected dangerous events and can export them in a format allowing a human operator to evaluate the dangerousness of the observed level crossing, calculate statistics and monitor the evolution of these events over time. It can also be used to evaluate the effectiveness of safety measures implemented on the level crossing, by monitoring the evolution of the number and type of dangerous behaviors that occur before and after the implementation of the measures.

The RES is organized in three consecutive stages. The first stage, called detection stage, takes as input the video data to analyze and perform the initial detection of the level crossing state and its users. Indeed, in order to classify dangerous behaviors related to the level crossing, it is necessary to extract information about the position of the barriers and/or the state of the warning lights. Users must be detected and classified into precise categories since the same behavior, mainly in terms of motion, may be classified as dangerous only if the user is a pedestrian instead of a road vehicle.
The results of the detection stage are then used as input by the second stage, i.e., the tracking stage. This stage reconstructs space-time trajectories based on the level crossing users detected in each frame of the input video during the first stage. It actually correlates the detection in one frame to the previous and next ones in order to have a full picture of the motion of a user from the first time it is detected to the last frame in which it is visible.

The final stage is the heart of the entire process. It is the recognition stage that takes as input the space-time trajectories built by the tracking stage and the state of the level crossing extracted by the detection stage. Those inputs are combined to recognize a chosen set of behaviors with various levels of dangerousness. It also uses simulation to compare the trajectories with ideal ones, i.e., the trajectory a user should have when strictly following the traffic code. This comparison is the basis of the anomalous behavior recognition that can be used to extract information about the categories of behavior for which no detector currently exists in the application, allowing developers to improve the system based on more complete information.

The complete architecture of the system is illustrated in Figure 2.1.

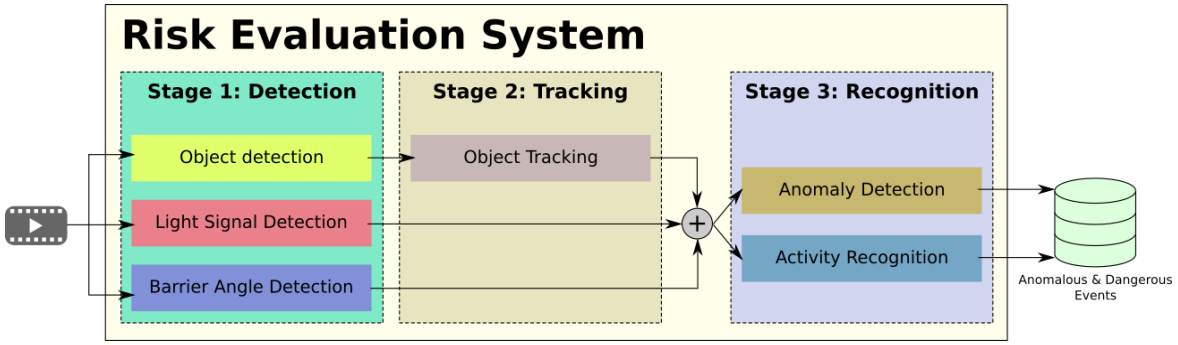


Figure 2.1 - Architecture of the Risk Evaluation System

2.2. Detection Stage

The detection stage is composed of operations designed to extract information about the state of the level crossing during each frame of the input video as well as information about the users. It consists of three main modules: light signal detection, barrier angle detection and object detection.

- The light signal detection consists in determining whether the light signals of the level crossing are active during each frame of the input video. Different light signal types may be used in different countries (single blinking light, two-state light, three-state light, etc.), therefore the detector was designed to handle all these configurations based on the input the RES user provides at the beginning of the process.
- The barrier angle detection consists in estimating the angle of each barrier of the level crossing in image-space, i.e., relative to the upward direction in the image. This estimation is performed by a deep learning model specifically trained for this task using a large dataset of synthetic images. The estimated angle is then mapped to a percentage of “openness” for each frame in order to be processed by the subsequent stages.
- The object detection consists in detecting static and dynamic objects, classifying and filtering them in order to keep only the relevant classes (pedestrian, car, bus, etc.). This operation is performed using a state-of-the-art deep learning detector called YOLO (Redmon et al. 2016).

Below we describe each module in detail.

2.2.1. Light Signal Detection

Light signal detection consists in determining whether the light signals of the level crossing are active during each frame of the input video.

To perform this operation, the RES user must input the rectangular region of interest (ROI) corresponding to each traffic light visible in camera reference frame. This ROI is constant for all the frames of the video as the camera is static.

For each ROI, the detector converts the pixel to HSV colorspace. The V values of each ROI in every frame form a discrete signal that is first normalized and is then processed in one of two ways depending on the type of light signal:

- if the signal is a two-stage or three-stage one, i.e., if the light is supposed to stay on as long as the level crossing is closing or closed, then the V value is simply compared against a user defined threshold. If the value exceeds the threshold, the signal is considered active, otherwise inactive.
- if the signal is a blinking light, the detector uses the Goertzel algorithm (Goertzel 1958), a well-known signal processing algorithm. It actually calculates the discrete Fourrier transform of a single frequency of the signal of the V value over time. The reference frequency is assumed to be known and should be provided by the RES user. The magnitude of the reference frequency in the signal is then tested against a user defined threshold. If the value exceeds the latter, the signal is considered active, otherwise inactive.
- The active state of each ROI in the input video is finally combined to form a single state signal for the whole level crossing using a logical OR operation. This simple rule for combining the signal states prevents the detector from not properly recognizing the state of the level crossing when a light signal is malfunctioning or is occluded by vegetation, vehicles, or other objects.

2.2.2. Barrier Angle Detection

In order to know the state of the level crossing, the system must be able to detect whether the barriers are closed or open (if the level crossing has any). The barrier angle detector of the RES is based on a deep learning model, namely ResNet50 (He et al. 2016). The model is specifically trained to estimate the angle of a barrier within a ROI, relative to the vertical direction of the image. In other words, the system estimates the angle of a barrier in image-space, the angle is then mapped to a fraction of the full angular range of the barrier.

ResNet50 is a deep residual neural network, i.e., a neural network that uses residual blocks with skip connections that jump over some layers in order to avoid the problem of vanishing gradients. This model is trained with our own dataset containing 60,000 images of barriers in various positions and under different lighting/weather conditions. The images of the dataset are equally distributed in 60 classes corresponding to the angles of the barrier in the image rounded to the nearest multiple of 3° from -87° to 90°.

Capturing a large dataset of real-world barriers pictures is a time-consuming process. The dataset should contain pictures under various lighting and weather conditions, which would require taking pictures at different times of the year, each picture must then be labelled manually. Instead, we chose to use synthetic images that could be automatically labelled as they are generated. To generate the dataset, a 3D reconstruction of a real level crossing and its surrounding was integrated in a simulation software developed in UTBM's CIAD Laboratory. This simulation software implements a physically based model of light scattering in the atmosphere (Elek and Kmoch 2010), and some of the algorithms from the Astronomical Algorithms (Meeus 1998) allowing the simulator to determine the actual position of the Sun in the sky of the level crossing at any time. The simulator also provides

a model for the cloud layer allowing the scene to be simulated under various weather conditions. Figure 2.2.2 shows some images from the dataset.

The images of the barriers were captured from various points of view in order to train a general model that could classify the barrier angles regardless of the position of the camera relative to the level crossing. Once the model returns an estimation of the barrier angle, it is post-processed to filter out obvious errors and to ensure coherency of consecutive angles. Let α_{closed} and α_{open}, respectively the angles of a barrier in fully closed and fully open position, with $\alpha_{closed} < \alpha_{open}$, $\alpha^\star$ the prediction generated by the model, the current barrier angle α_t is calculated using the following equation:

$$\alpha_t = \begin{cases} (\alpha^\star - \alpha_{mid} + 90) \pmod{180} + \alpha_{mid} - 90, & \text{if } |\alpha^\star - \alpha_{t-1}| \leq 10 \\ \alpha_{t-1}, & \text{otherwise} \end{cases}$$

with:

$$\alpha_{mid} = \frac{\alpha_{closed} + \alpha_{open}}{2}$$

Incoherencies are caused by the fact that the detector does not differentiate between the tip and the base of the barrier and returns angles in the range -87° to +90°. If we trained the model in the range -180° to +180°, two angles would be valid for the same image, i.e., the actual angle and its opposite. Therefore, fine tuning is expected to avoid these ambiguities under the premise that the angle of the barrier cannot change greatly during a short period of time. An alternative would be to train the model to recognize the tip and the base of the barrier. This solution, however, requires the RES user to select a large ROI that includes the full barrier, while our proposed solution only requires a small ROI in which the dashed pattern on the barrier is visible.

In order to smooth the angles over time and to interpolate values that are missing when the estimation is way off, a simple Kalman Filter (Kalman 1960) is used to track the angle of each barrier.

Figure 2.2.2 - Some images from the barrier angle dataset

2.2.3. Object detection

To recognize dangerous behaviors, the system must be able to detect the users of the level crossing. This operation is performed by the object detector based on a deep neural network model called YOLO (Redmon et al. 2016). YOLO[2] is the state-of-the-art object detector and classifier consisting of a single neural network applied to the whole image divided into regions and predicts bounding rectangles, classes and probabilities for each region. The bounding rectangles are weighted by the predicted probabilities. Figure 2.2.3 shows a typical output of the model in our RES.

[2] https://pjreddie.com/darknet/yolo/

In this work, we used the version 3 of the model pre-trained on the COCO dataset[3]. The output of the detector is filtered to only keep the classes that are relevant, i.e., classes that correspond to level crossing users (pedestrian, bicycle, motorbike, car, truck and bus).

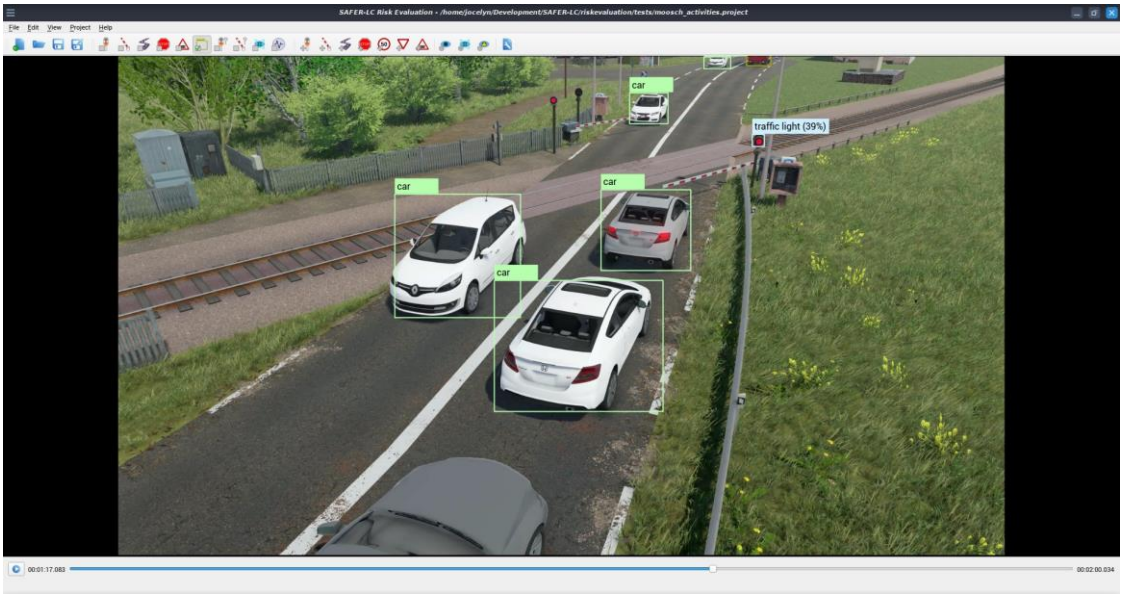


Figure 2.2.3 - Example of object detection output

2.3. Tracking Stage

The tracking stage is responsible for converting the discrete level crossing user detections for each frame produced by the first stage into coherent space-time trajectories for each user. This operation requires associating detection of an object across multiple frames. The tracker must overcome several challenges / uncertainties:

- a user may not be correctly detected in all the frames in which it is visible,
- a user may not be properly categorized,
- elements in the background may be mistakenly detected as a user,
- bounding rectangles may not match the actual shape of the user in the image.

In order to track an object, a reference position must first be selected. We chose the centroid of the bounding rectangle of a detected user since its position is most of the time stable with respect to the shape of the object.

The classical approaches for object tracking require using the reference positions of the objects in one frame and predicting their positions in the following frames. A Hungarian algorithm-based data association algorithm is used to associate detections corresponding to the same physical object across frames.

[3] http://cocodataset.org/

Tracking can be performed in image-space, or scene-space (i.e., the local 3D coordinate system of the scene). We chose to implement our tracking in the latter while ignoring the Z (vertical) coordinate in order to reduce computational complexity. Indeed, all level crossing user motions are restricted to the ground plane, which we assume to be relatively flat in the surrounding of the level crossing. Also, tracking an object in image-space can be challenging since the speed of the object may dramatically depending on the position in the image due to the perspective effect of the camera. Tracking objects in scene-space, however, requires the reprojection of their centroids from image-space to scene-space, which necessarily requires information about the depth, in other words the distance between the camera and the object. This information is usually obtained using stereoscopic camera which records the RGB component of each pixel and estimates its depth. Since the RES makes no assumption about the type of camera used to record the input video, it cannot rely on the depth information. Instead, centroids are projected onto a virtual plane above ground whose height depends on the category of the object. The virtual plane corresponds to the plane in which all centroids of objects in the same category are located. For example, centroids of cars are located at around 80cm above ground, trucks at around 1.5m, pedestrians at around 90cm, etc. Indeed, this reprojection introduces errors in the position of the object, which are added to the error of the position of the centroid in image-space (due to incorrect bounding rectangle) and the error possibly introduced by a mis-calibration of the camera. However, the categorization of the motion of a user in SAFER-LC project is mostly relative to the rail and road infrastructure. We therefore assumed, that the impact of the introduced errors is small enough to be ignored. Our tests confirmed this hypothesis. One limitation of our approach is that the error in reprojection is dependent on the quality of the classification, if a car is mistakenly detected as a truck, the virtual plane will be 70cm too high, which may introduce a large error at grazing angles.

Our tracker uses a Robust Adaptive Unscented Kalman Filter (Zheng et al. 2018) to predict the position of the centroid of each object in the next frame. The main advantage of this type of filter is that it automatically adjusts the covariance matrices based on the provided observations. It makes the filter very robust against dynamic noise. The state vector is defined as:

$$\begin{pmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{pmatrix}$$

where x and y are the components of the centroid position in the virtual plane and $\dot{x}$ and $\dot{y}$ are the components of its velocity.
The state transition matrix is simply defined as:

$$\begin{pmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Data association is performed using the Hungarian algorithm that finds the cost minimizing assignment. For every tracked object i and observation j, a cost matrix is constructed using the following rules:

- if the confidence factor for the detection of observation $j$is lower than a user specified threshold, the cost is set to an arbitrary large value,
- if the distance in scene-space between the observed position $\boldsymbol{p_j}$and the position $\widehat{p_i}$predicted by the Kalman filter is larger than a user specified threshold, the cost is set to an arbitrary large value,
- the base cost is the image-space squared distance between the observed position $\boldsymbol{p_j}$and the position $\widehat{p_i}$predicted by the Kalman filter,

if the category of observation $j$is different of the category of the tracked object i, the base cost is squared.

The first two rules are used to ensure that obviously invalid assignments cannot be selected by the algorithm. Indeed, if an invalid assignment is selected, it means that two tracks for two different objects may be merged into a single one. For each tracked object, if an observation has been assigned by the Hungarian algorithm, it is used to update the Kalman filter. Otherwise, the Kalman filter uses its predicted position to estimate the state of the object in the next frame. Objects for which no observation has been assigned for a user-defined number of consecutive frames are stopped and all predicted states since the last observation are removed. Unassigned observations are used to create new tracks.

When an observation is added to an active track, the system tries to infer the most probable class of the object by selecting the class for which the sum of all scores is the highest. If the new class differs from the one the track previously had, the system recalculates the whole track while reprojecting the observations on the virtual plane associated with the new class. This, of course, slows down the tracking algorithm significantly but it does improve accuracy and reduces fragmentation caused by the misclassification of an object.

At the end of the process, only tracks longer than a user-specified threshold are kept and post-processed. The post-process consists of applying a low-pass filter to remove noises introduced by reprojection errors and incorrect bounding rectangles. We used the Savitzky-Golay (Savitzky and Golay 1964) filter to smooth the tracks by fitting successive subsets of adjacent points with a low-degree polynomial using the least-squares method. The window size and the degree of the polynomials are adjusted on a per category basis. Indeed, since pedestrians and road vehicles do not have the same movement constraints, their trajectories may be better fitted by polynomials of different degrees. Also, as the average speed of the pedestrian is unlikely to be close to the average speed of road vehicles, different window sizes for the filter need to be used.

Figure 2.3. shows an example of trajectories reconstructed by our system.

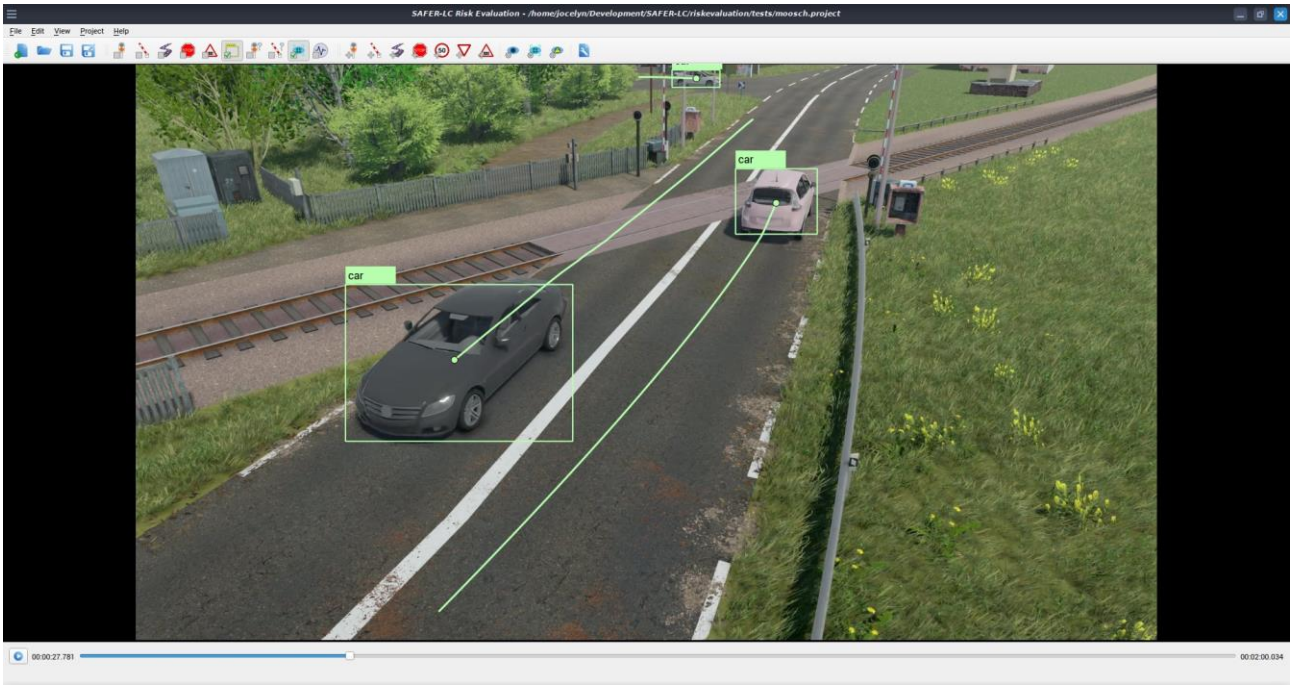


Figure 2.3 - Example of multi-object tracking output

2.4. Recognition Stage

The recognition stage is the final stage and the key of our system. It consists of using information about the state of the level crossing extracted during the detection state, trajectories generated by the tracking stage, and information about the road and rail infrastructure provided by the RES user, in order to detect some specific behaviors. Additionally, the recognition stage can also flag each trajectory that is considered to be anomalous. This anomaly detection is intended to help maintainers of the system improve the application by creating new types of detectors based on the observed anomalous behaviors.

2.4.1. Anomaly Detection

Anomaly detection is performed by comparing the observed behavior of a level crossing user, through its trajectory, to a reference normal behavior. It can only be used for road vehicles because of difficulties in simulating the behavior of a pedestrian without knowing its goals, motivations and internal (i.e. mental) state.

The reference normal behavior for a road vehicle is generated by replaying the scene in simulation while replacing the observed user with a virtual driver that possesses an artificial intelligence that strictly respects the traffic law.

2.4.2. Simulator architecture

The simulator used to generate the reference trajectories has an architecture similar to the one used to generate most of the videos used to test the RES. It is based on the environment model described in (Buisson et al.). This model, represented as a graph of interconnected navigable zones in 2D, allows the virtual driver to perceive and act in its environment. In the context of this work, these navigable zones represent the lanes of the road network. Navigable zones are connected through two different kinds of links: navigation link (i.e. explicit link) and perception link (i.e. implicit link).

The navigation link connects two geometrically adjacent navigable zones allowing a vehicle to move from one to the other. While the perception link connects two overlapping navigable zones allowing a vehicle located on one zone to perceive objects located on the other. Lanes are geometrically modelled as sets of convex polygons constructed from extrusion along the normal of a discretized cubic Bézier curve. Figure 2.4.1.1.a illustrates the two steps in the construction of a lane, i.e., lateral extrusion then discretization (sometimes called flattening). This modelization is used to speed up intersection tests. To reduce even further the computational complexity of the simulation, the polygons of the lane are stored in a R-Tree (Guttman and Stonebraker 1983).

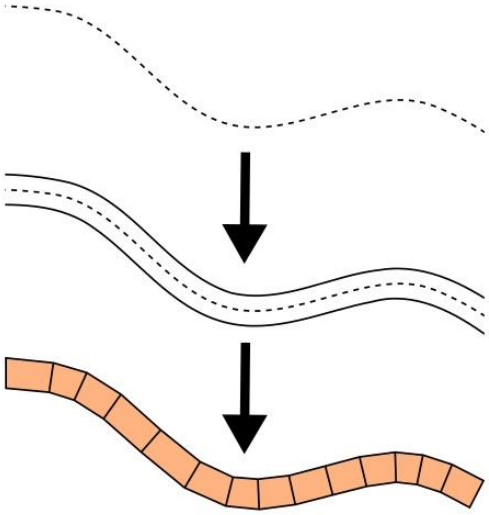


Figure 2.4.1.1.a - The two steps involved in the construction of a lane from a Bézier curve

Lanes hold information about all the perceptible objects (that we refer to as “entities”) that intersects their geometry. Each lane belongs to a “road segment” allowing a driver to access information about adjacent lanes in order to plan lane-changing maneuvers.

The road segments are collections of adjacent lanes sharing the same direction, bounded by two “road connectors”. Using this architecture, a two-way street must be modelled using two opposite road segments. This prevents vehicle on one side of the road to perceive the vehicles on the other since there is no connection between the segments, and therefore, forbid overtaking maneuver. Although it could be a problem if our objective was to simulate general traffic on rural roads, this

does not affect our work since overtaking is strictly forbidden in the vicinity of a level crossing and the simulator only contains law-abiding drivers.

The road connectors are elements that bounds road segments and allow them to be connected to construct a network. The connector holds information about all incoming and outgoing connections.

An incoming connection represents the connection of a road segment that uses the connector as its end point while an outgoing connection represents a connection with a road segment that uses the connector as its starting point. The connections also contain an offset providing flexibility in the way the lanes of road segments are connected. For example, suppose a connector contains two connections, an incoming one with a road segment **A** that has three lanes, and an outgoing connection with an offset of 1, and with a road segment **B** that has two lanes. This means that the left-most lane of segment **B** will be connected to the middle lane of segment **A**, as illustrated in Figure 2.4.1.1.b.

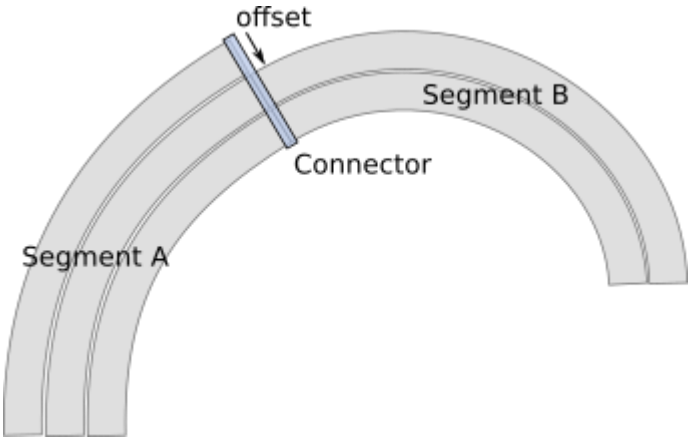


Figure 2.4.1.1.b - Connection with offset between two road segments.

The entities are perceptible objects in the simulation. Each entity is modelled as a convex polygonal object, usually an oriented rectangle. There are three types of entities in our simulation: the animated road users, the vehicle of the simulated virtual driver (artificial intelligence with autonomous behavior), and the traffic signs.

The animated road users are objects for which a trajectory has been generated in the tracking stage, excluding the one replaced by the virtual driver. At each step of the simulation, their position is updated to the one contained in their trajectory at the corresponding frame, along with their orientation updated to match the direction of the velocity vector contained in the trajectory. The animated road users are associated with a time of arrival and are added to the simulation several seconds before their track actually begins, the actual number of seconds is a parameter of the anomaly detection phase called “anticipation time”. This is necessary in order to take into account the fact that the recording camera has a limited range and cannot cover all entry points of the level crossing at the same time. Without this artefact, a virtual driver waiting to cross an intersection would not have information about an oncoming vehicle that is outside the field of view of the camera and would introduce bias in the simulation, which prevents it from predicting the reference normal behavior.

The vehicle of the virtual driver is moved using a standard bicycle model. At each step of the simulation, the virtual driver calculates the steering angle, i.e., the mean angle of the steered wheels and the longitudinal acceleration of the vehicle. The new position and velocity vector is then calculated using the bicycle model.

Our model contains four types of traffic signs, namely speed limit, yield, stop, and level crossing. The speed limit sign is straightforward and only contains the maximum allowed speed beyond its position. The yield and stop signs contain information about possible directions. A direction is defined as a lane beyond the sign toward which a vehicle can navigate. It is used to differentiate between left and right turns at an intersection. Each direction is associated with a set of information for each conflicting lane:

- the length of the conflict zone, i.e., the length of the overlap of the conflicting lane along the direction lane,
- a position from which a driver is supposed to look backward in the graph for a potential incoming vehicle that would prevent him from passing the sign,
- a flag that indicates whether queuing in the conflict zone is allowed (for right-handed traffic queuing in the conflict zone when turning right is allowed).

The level crossing signs simply hold the current state of the level crossing. They are placed on incoming lanes just before the level crossing polygon input by the RES user using the interface described in section 2.5.5.

The road network is constructed by the RES user using the interface described in section. Automatic construction of the road network takes place once the lanes are input, as detailed below:

1. The perception links are added to connect each couple of overlapping lanes that do not share a connector.
2. If the RES user inputs the level crossing zone, a level crossing sign is placed on each lane that overlaps with this zone and leads to it.
3. The speed limit signs are placed on their respective lanes.
4. The yield signs that model priorities are automatically placed using the following rules:
    a. When two lanes overlap, a yield sign is placed on the lane that does not have the right of way, just before the conflict zone.
    b. All paths through the conflict zone are calculated and a direction is created for each lane that exits the conflict zone.
    c. The length of the conflict zone is calculated by projection along the central Bezier curves of the lane in each path.
    d. The queuing flag is set when the conflict represents a merging, i.e., when the conflicting lanes share the same end connector, otherwise it is unset.
5. The yield and stop signs manually placed by the RES user through the interface of the application are then added using a procedure equivalent to the previous one except for the following aspects:
    a. When the sign is placed on a lane that had previously the right of way, all yield signs related to the conflict placed on each conflicting lane are removed.
    b. When the sign is a stop and is placed on a lane that had not previously the right of way, the automatically placed yield sign is replaced.

2.4.3. Simulation cycle

The simulation for anomaly detection starts at the first frame of the video in which the observed user is present. The vehicle, replaced by an entity controlled by the virtual driver is placed at the first position in its trajectory, its orientation is set to match the direction of the initial velocity vector in the trajectory. The simulation cycle is then composed of the following steps:

- the positions of all animated road users are updated to match the position defined in their trajectory at the current frame,
- each lane of the road network is updated accordingly, i.e., the collection of all entities that intersect their geometry is reconstructed,
- the virtual driver is executed, using the data-structure of the environment it can perceive obstacles and traffic signs, makes a decision on which action to take and compute the steering angle and acceleration his vehicle must have for the current step,
- the position and orientation of the virtual vehicle is computed using the bicycle model.

Since some parameters that affect the behavior of the virtual driver are randomly selected, the whole simulation cycle must be repeated several times in order to produce a behavior that matches the real one as closely as possible. Only the closest “ideal” trajectory is kept at the end of the process.

2.4.4. Virtual driver behaviour

The virtual driver behavior is composed of three main parts, namely longitudinal control, lateral control, and traffic sign handling.

The longitudinal control part is responsible for managing the acceleration and speed of the driver in accordance with the context and traffic laws. This component takes as input three elements: the short-term trajectory of the vehicle, the most immediate obstacle in front of the vehicle and the current speed limit. Given these data, the longitudinal controller must first estimate the maximum curvature κ_{max} of the short-term trajectory in order to adjust the speed of the vehicle in anticipation. Maximum trajectory curvature is estimated using a recursive procedure based on the Frenet–Serret formula:

$$\frac{d\mathbf{T}}{ds} = \kappa\mathbf{N}$$

Where $\mathbf{T}$ is the tangent vector of the trajectory, s is the arc length, $\mathbf{N}$ is the normal vector of the trajectory and κ is the curvature of the curve as illustrated by the figure 2.4.1.3.

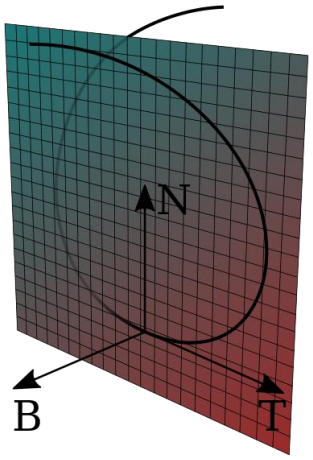


Figure 2.4.1.3 - The tangent-binormal-normal (TBN) frame of a point on a curve (from Wikipedia), we do not use the binormal vector B since the model of the environment is in 2D

Using the magnitude of the $\mathbf{T}$ and $\mathbf{N}$ vectors, the equation can be rearranged as:

$$\kappa = \frac{||\frac{d\mathbf{T}}{ds}||}{||\mathbf{N}||}$$

In order to approximate κ, a look-ahead distance l is calculated as:

$$l = max(l_{min}, t_l \cdot u_v)$$

With $l_{min}$is the minimum look-ahead distance, t_l is the look-ahead time and u_v is the longitudinal speed of the vehicle.

Assuming the arc length of the vehicle's reference position on the curve is s_v, a first κ is computed for the section of the curve between s_v and $s_v + l$. Simultaneously, a circle is fitted between the current position of the vehicle and the position at the look-ahead distance on the curve using the tangent of the curve at the position of the vehicle as additional constraint. If the curvature of the trajectory obtained with the Frenet–Serret is not close enough to the curvature of a fitted circle, the trajectory is split at the position defined by the arc length $s_v + \frac{l}{2}$ and the process is repeated on each part of the curve until either the ratios between curvatures are sufficiently close to 1 or the length of the curve part to evaluate becomes too small. The maximum curvature is the maximum of all κ for each part of the curve.

Once the maximum curvature is estimated, the longitudinal controller can compute the maximum speed at which the vehicle can go by specifying a constraint on the lateral acceleration of the vehicle. Lateral acceleration, also called centripetal acceleration, is a parameter of the behavior that defines the tolerance of the vehicle driver and its passenger to lateral G-forces. Lateral acceleration a_c is related to the speed of the vehicle and the curvature by the following equation:

$$a_c = \kappa \cdot u_v^2$$

In order to ensure the driver does not accelerate in the middle of the curve, when the estimated maximum curvature decreases, a maximum speed is computed with the same equation, but using the curvature of the current turning circle instead of the curvature of the trajectory. Knowing the steering angle δ of the vehicle, i.e., the mean angle of the steered wheels, the curvature of the current turning circle κ_s is determined using the following equation:

$$\kappa_s = \frac{tan(\delta)}{E}$$

Where E is the wheelbase of the vehicle. The maximum speed v_{max} at which the driver should drive is therefore taken as the minimum between the maximum speed for the current trajectory, the maximum speed for the current steering angle and the current speed limit.
The free-driving acceleration i.e. the acceleration the vehicle should have, if there is no obstacle in front of it, is then calculated using the following equation:

$$a_{free} = \begin{cases} a_{pref} \cdot q, & \text{if } q \geq 0 \\ d_{pref} \cdot -q, & \text{otherwise} \end{cases}$$

with:

$$q = 1 - (\frac{u_v}{v_{max}})^e$$

where a_{pref} and d_{pref} are respectively the preferred acceleration and deceleration of the driver and e is the acceleration exponent that controls the steepness of the acceleration curve.
In case an obstacle blocks the path of the vehicle, whatever it may be (e.g., another vehicle, a traffic sign or an object on the road, etc.) the longitudinal controller calculates the car-following acceleration a_{follow}, i.e., the acceleration the vehicle should have to avoid colliding with the obstacle and maintain a safe distance. Many car-following models could be used to calculate this acceleration like the well known Intelligent Driver Model (Treiber, Hennecke, and Helbing 2000) or the Gipps' model (Treiber, Hennecke, and Helbing 2000; Gipps 1981).
We used the car-following model described in (Perronnet 2015), which is defined by the following equation:

$$a_{follow} = \frac{b_f\tau - 2v_f - 2b_f\sqrt{\frac{b_f b_l \tau^2 + 4b_l b_f \tau + 4{v_l}^2 - 8b_l(s-s_0)}{4b_f b_l}}}{2\tau}$$

Where v_f and v_l are respectively the current speed of the vehicle and that of the obstacle in front, s is the headway, s_0 is the minimum separation distance between the vehicle and the obstacle when the vehicle is stopped, b_f is the preferred deceleration of the driver, b_l is the maximum (supposed) deceleration of the obstacle and τ is the reaction time of the controller.

The lateral control part of the driver behavior is responsible for computing the mean steering angle of the vehicle in order to follow the short-term trajectory. The literature contains many such models like the classical pure pursuit model (Carnegie-Mellon University, Robotics Institute and Craig Coulter 1992). One drawback of the original pure pursuit model is that vehicles tend to "cut corners" at sharp curves when using a large look-ahead distance.

We used our own lateral control model that is based on curvature tracking and uses a pure pursuit part to correct path tracking error over-time. Our approach is to follow the curvature of the path as closely as possible in order to reduce drift. This can be achieved by aligning the steered wheels with

the tangent of the trajectory at the projected position of the center of the steered axle. Of course, some drift will necessarily occur since our simulation is executed at discrete time steps and the sampling rate may not be sufficient when the vehicle moves at high speed.

To correct the angular and lateral errors caused by drifting, we use a variant of the pure pursuit. Instead of searching for a look-ahead position on the trajectory itself, we select a point P_t on a line parallel to the tangent of the curve at the reference position of the vehicle projected onto the curve P_p that passes through this point. Figure 2.4.1.3. illustrates the pure pursuit correction.

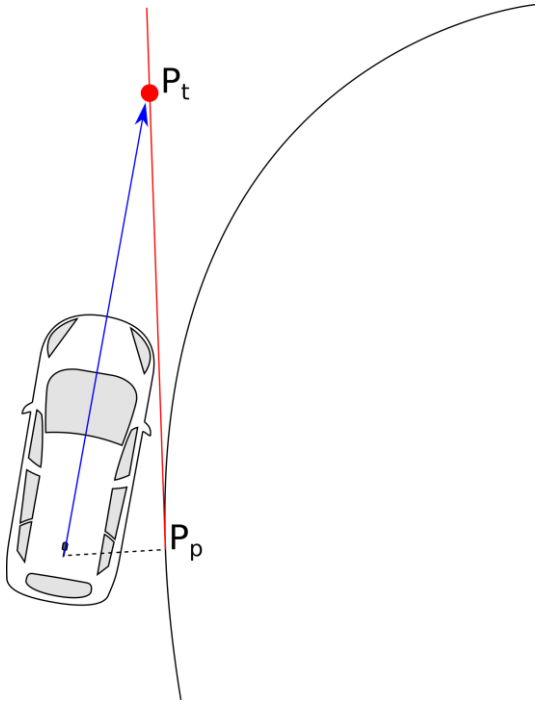


Figure 2.4.1.3 - Lateral error correction with pure pursuit

Except for the selection of the target point to follow, the pure pursuit algorithm is used in its original formulation. The choice of the pure-pursuit look-ahead distance only affects the speed at which the driver will correct path tracking errors and can be randomly selected to generate a multitude of different behaviors while still ensuring that all vehicles will take sharp turns properly.

The final steering angle to apply to the vehicle is simply the sum of both angles: the angle formed by the direction of the vehicle and the tangent to the curve at the projected position of the center of the steered axle and the pure pursuit correction angle.

The traffic sign handling part of the behavior is responsible for interpreting traffic signs for the driver, changing the current speed limit and creating virtual obstacles when the driver should stop before a sign.

Our simulator implements only four kinds of traffic signs: speed limit, stop, yield and level crossing. We voluntarily excluded the traffic lights because simulating them would require modifying the RES's architecture in order to extract information about the cycles of the traffic lights around the level

crossing. This would require multiplying the number of cameras that monitor the level crossing in order to watch the states of all traffic lights in the vicinity.

- The speed limit sign is very straightforward, if the current speed of the vehicle exceeds the speed limit of the sign, a virtual obstacle is created at the position of the sign with a speed equal to the limit, thus forcing the driver to decelerate until it reaches the correct speed. As we have shown previously, the longitudinal controller will make sure the vehicle decelerates smoothly until it is very close to the sign.
- The stop and yield signs are practically equivalent except that the driver is forced to stop before the stop sign. Force both of these signs, the driver checks the conflicting lanes associated with its direction for an oncoming vehicle that would have the right of way. If there is, the driver estimates its time to exit the conflict zone associated with the sign using its current velocity, preferred acceleration and the length of the conflict zone stored in the sign object. Based on this information and the known distance between the oncoming vehicle and the conflict zone as well as its current speed, the driver calculates the deceleration that the oncoming vehicle should have to avoid colliding with itself if it were to cross the conflict zone at that time. If this deceleration is less (in absolute value) than a specified threshold, the driver will cross the sign, otherwise a virtual obstacle is created in place of the sign to force the vehicle to stop or, at least, decelerate. This threshold that we name *minimum imposed deceleration* is analogous to courtesy and can be randomly selected (in a reasonable interval) to generate various driver behaviors.
- The level crossing sign is an artefact that indicates to the driver the position on the road where it should stop when the level crossing is closed or closing. As long as a level crossing sign is in the forward perception of the vehicle driver, its status is checked and a virtual obstacle is created at the exact position of the sign if its warning lights are on or if the barriers are closing or closed. Practically, a level crossing sign is handled similarly to a regular traffic light in the simulation.

2.4.5. Activity Detection

Activity detection is the last component and the heart of the RES. Based on the extracted trajectories of level crossing users and information about the rail and road infrastructure provided by the RES user, a set of rules, explained in the following sections, is used to detect some activities which may be dangerous. We focused our development on detecting activities related to the crossing of the level crossing. The system is able to detect 7 kinds of potentially dangerous crossings: zigzagging, wrong-way crossing, illegal lane-changing, abnormal crossing, illegal crossing, stopping and queuing. Additionally, the system is also able to detect speeding of a road vehicle even if it is not actually crossing the rails.

All crossing-related activities share the same basic principle: the detector searches for the lane on which the user is at the moment it enters the level crossing zone as well as the lane on which the user is at the moment it leaves the zone. Based on the direction and connectivity of the road segment lanes input by the RES user, it is possible to infer the type of behavior the level-crossing user acted out while crossing.

2.4.6. Zig-zagging activity

A zig-zagging activity is detected when the user enters the level crossing on a lane with an opposite direction to its motion and when it leaves the level crossing on a lane that is directed toward its motion. Additionally, no valid path must exist between the entry and exit lanes. Figure 2.4.2.1 shows examples of zigzagging detected by our system.

Figure 2.4.2.1 - Zig-zagging activities detected by the system

2.4.7. Wrong-way crossing activity

A wrong-way crossing activity is detected when the user enters and leaves the level crossing on lanes with opposite direction to its motion. Additionally, there must exist a direct and valid path between the entry and exit lanes. Figure 2.4.2.2. shows examples of wrong-way crossing detected by our system.

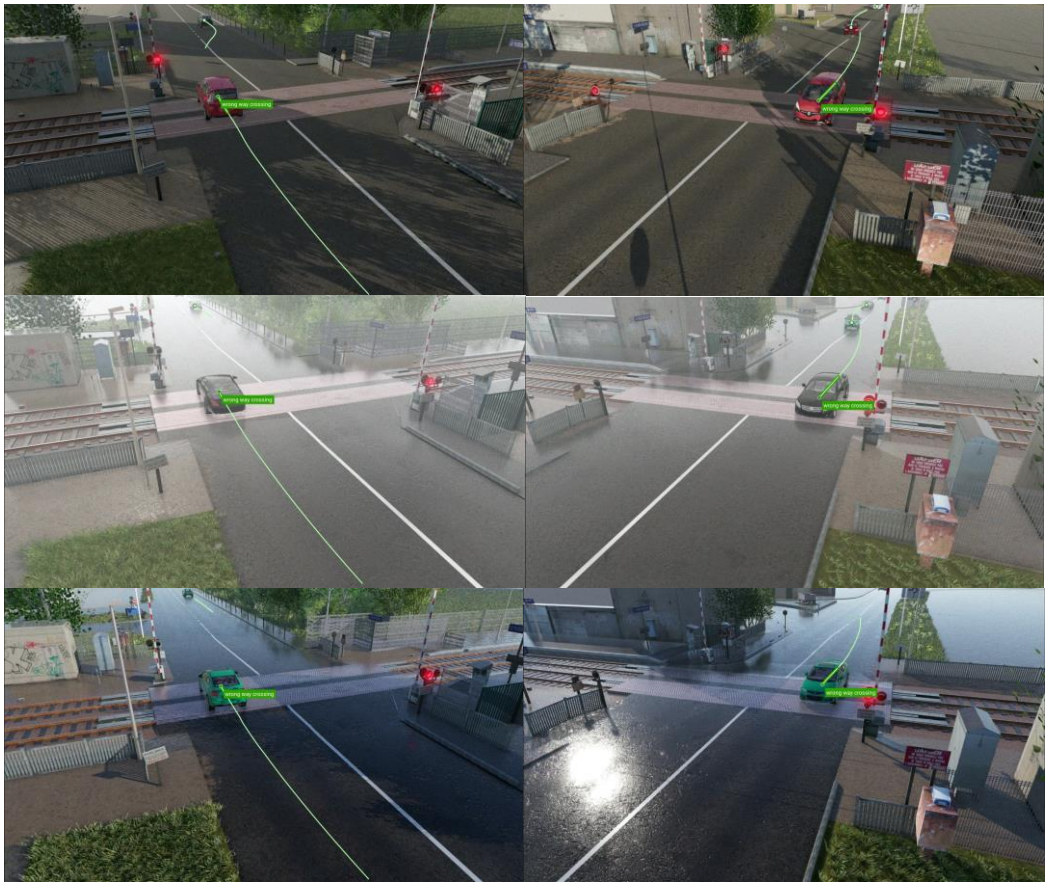


Figure 2.4.2.2 - Wrong-way crossing activities detected by the system

2.4.8. Illegal lane changing activity

An illegal lane changing activity is detected when the user enters and leaves the level crossing on lanes that have the same direction as its motion but there is no direct path between these lanes, however, there exists a lateral path between them, i.e., the lanes traversed by the user while crossing are adjacent.

2.4.9. Abnormal crossing activity

An abnormal crossing activity is the fallback activity that is triggered when the crossing of the user matches neither a normal crossing activity nor any of the previously defined activities. It could mean that the user entered the level crossing then stopped and backed up in order to leave the zone or the user performed an “inverse” zig-zagging, i.e., entered on a valid lane then exited the zone on a lane with an opposite direction.

2.4.10. Illegal crossing activity

This activity represents the crossing of the level crossing by any kind of user while the light signals are turned on and/or the barriers are closing or closed. This activity can be detected simultaneously

with any of the previously described activities. Figure 2.4.2.5 shows examples of illegal crossing detected by our system.

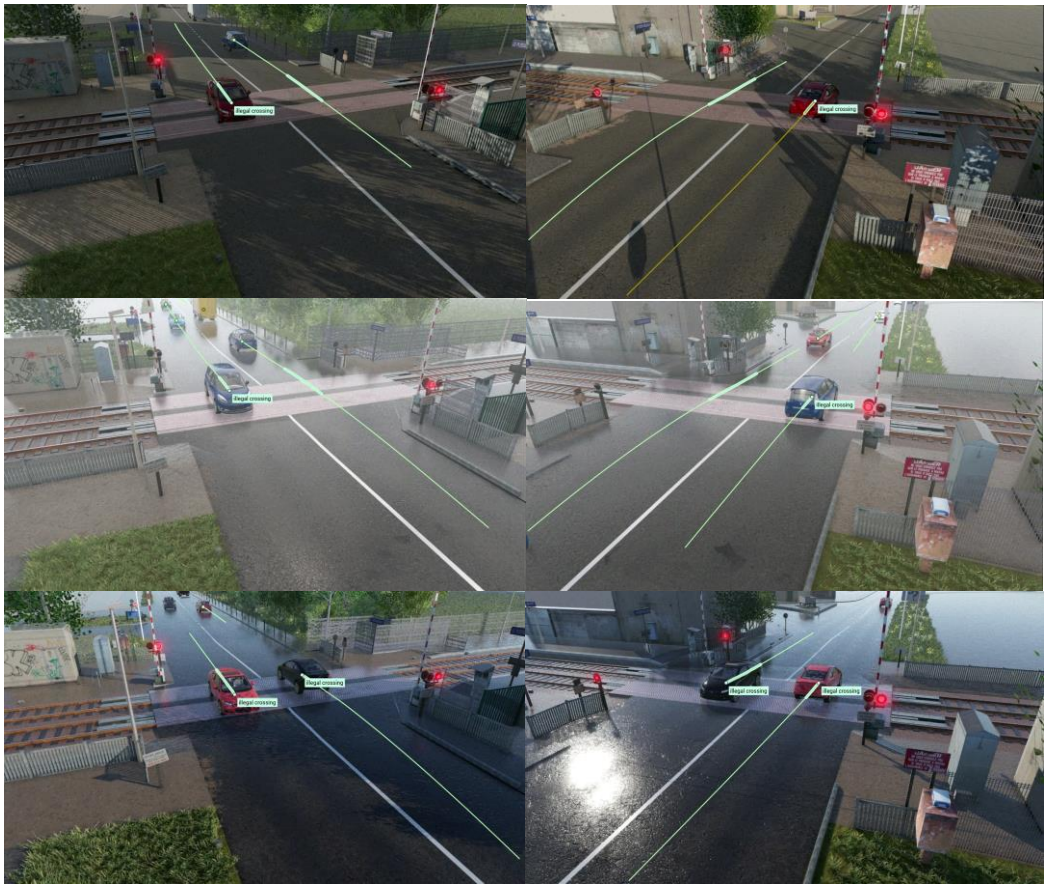


Figure 2.4.2.5 - Illegal crossing activities detected by the system

2.4.11. Stopping activity

A stopping activity is detected when a user takes more than five seconds to cross the level crossing. Note that unlike the queuing activity, the reason for the slow crossing cannot be inferred by the system. If the user is a vehicle this activity would be detected when the vehicle stops because of stalling or a mechanical malfunction. It could also mean that the vehicle stopped because of an obstacle on the road that cannot be detected by the system. Figure 2.4.2.6 shows examples of stopping activities detected by our system.

Figure 2.4.2.6 - Stopping activities detected by the system

2.4.12. Queuing activity

A queuing activity is similar to a stopping activity. It is triggered when a user takes more than five seconds to cross but unlike the stopping activity the system detects the reason for the slow crossing. Using the road network data structure that serves as a basis for the simulation for anomaly detection plus the tracks of all detected objects, the system can fill up the data structure with the known position of all objects at the point in time corresponding to the crossing. The system then searches in front of the user responsible for triggering the detector by locally traversing the graph of lanes and looks for an object that is distant of less than 25 meters (we assumed that most, if not all, drivers would close the gap if the distance was greater in order to avoid being stuck on the LC). If such an object exists, the activity is recorded.

2.4.13. Speeding activity

A speeding activity, while simple in principle, requires the system to perform a lot of preprocessing in order to assign for lane of the road network the portions associated with specific speed limit provided by the RES user. While checking the trajectory of an object on the road, the detector looks for a frame in which the velocity vector's magnitude is greater than the speed limit at that position. A

tolerance of 5 km/h is added when the speed limit is lower than 100 km/h and 5% when the limit is above 100 km/h in order to account for errors in the tracking process (this tolerance is the standard for speed cameras). If one such frame is detected, the system will then look forward in time for the first frame in which the vehicle is not speeding. If the time elapsed between the first and the last speeding frame is greater than a user specified threshold (we chose half a second as a default), the speeding activity is recorded. Otherwise, the system checks if the duration of the non-speeding period is greater than or equal to the threshold. If the non-speeding duration is too short, the system acts as if the user was speeding the whole time. This method of detecting speeding activities provides more robustness against short periods of speeding and non-speeding, which may be caused by tracking errors.

2.5. User Interface

We developed a graphical user interface for the RES in order to make the process user friendly. In this section we detail the different tools and dialogs of the application and describe how to use the system. This user interface is developed with the version 5 of the Qt framework4 (a cross-platform GUI API).

2.5.1. Main interface

The main interface of our RES looks similar to a video player software. Figure 2.5.1. shows the main interface and is components: it is composed of a central component (1) onto which the video is displayed as well as overlays for the output of each stage of the process. The bottom part (2) is occupied by a timeline component and a status bar that displays information when a button or a menu is hovered or when an editor action is in progress. The upper part of the main window has a menu bar (3) that provides access to all the features of the application and a set of toolbars (4) that provides a faster access to those same features.

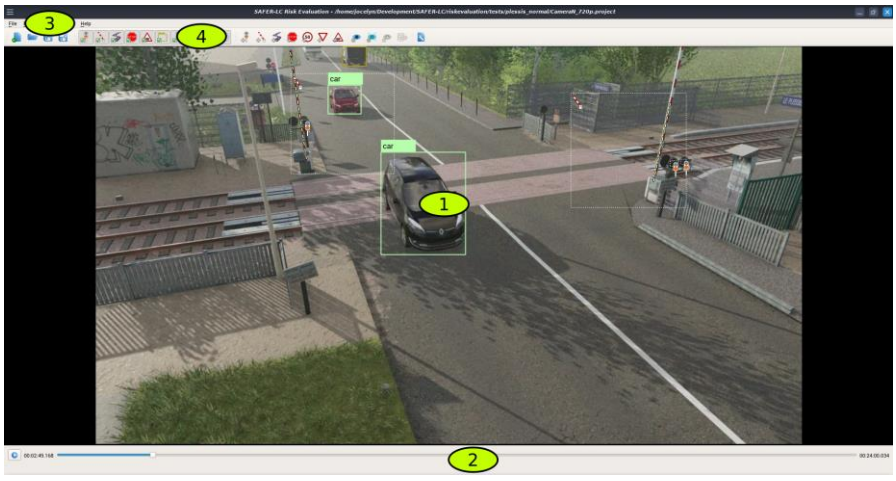


Figure 2.5.1 - The main interface of the RES

[4] https://www.qt.io/

The application responds to keyboard shortcuts like the space bar that play/pause the video, arrow keys move the position in the timeline. Also, standard keyboard shortcuts like save/save as are also recognized.

In order to allow the RES user to input the necessary information about the road and level crossing infrastructure, a set of editors was implemented and are described below.

2.5.2. Light signal editor

The light signal editor allows the RES user to specify the ROI of each light signal visible in the video. To create a light signal ROI, the RES user must first click on the toolbar button with the following icon  then click and drag the mouse cursor onto the video. Once a ROI is created, it can be modified by first selecting it with a click and then by moving the handles displayed on the edges of the ROI. Figure 2.5.1.1. shows two light signal ROIs, the left one is selected and the editor handles are visible.

Figure 2.5.1.1 - Light signal editor in the RES application

2.5.3. Barrier editor

The barrier editor allows the RES user to specify the ROI of each barrier as well as its open and close pose in the camera reference frame. Similarly to the light signal editor, to create a barrier ROI, the RES user must first click on the toolbar button with the corresponding icon  then click and drag the mouse cursor onto the video. The RES user must then move the pivot of the angle gizmos and adjust the open and closed handles as illustrated by the figure 2.5.1.2.

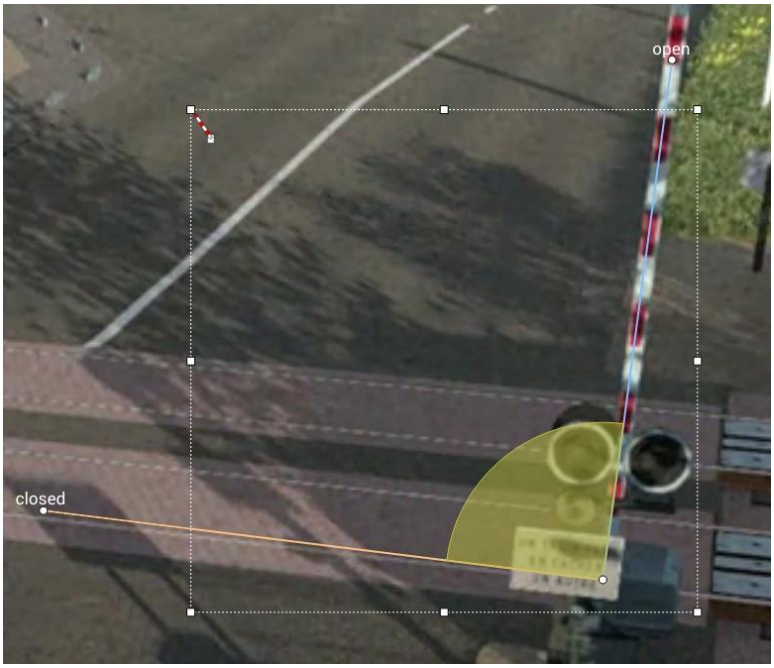


Figure 2.5.1.2 - Barrier editor in the RES application

2.5.4. Road network editor

The road network editor is composed of two elements: the road segment editor and the traffic sign editor. The former is used to draw road segments and connect them to create the network, while the second one is used to place traffic signs onto the appropriate segment.

To create a road segment, the RES user must first click on the toolbar button with the corresponding icon  then click and drag onto the video. The road segment will start at the position the mouse button was pressed and end at the position the mouse button was depressed. Since road segments are modeled using cubic Bézier curve, it is often necessary to adjust the direction and length of the tangents. When a road segment is selected, a set of handles are displayed on the gizmos representing the segment as illustrated by the figure 2.5.1.3.a. Handles in the middle of the segment at the beginning and end can be used to move the control points. Handles on lateral extremities of the segment can be used to change the width of the segment which is interpolated along the curve. Tangents are displayed using a dashed line and the handle at the extremity of this line can be used to modify both the direction and length.

In order to be displayed, the cubic Bézier curve of a road segment is first projected onto the ground plane in the scene frame, it is then extruded, discretized as a polygon, then projected back to the camera frame.

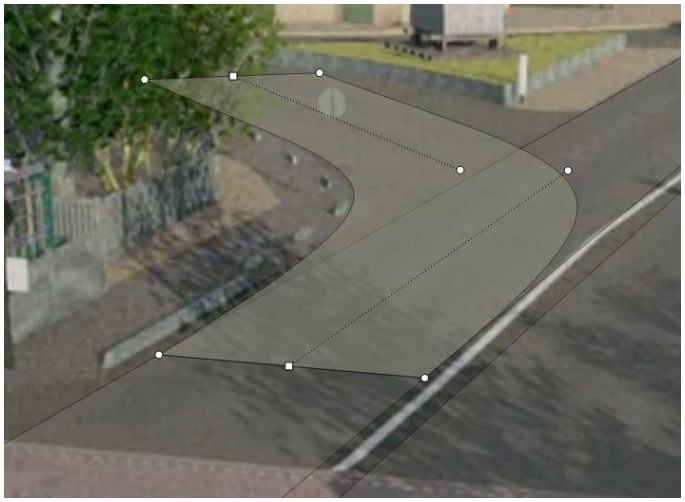


Figure 2.5.1.3.a - Road segment editor in the RES application

The traffic sign editor is quite simple in comparison. To create a traffic sign, the RES user must select the type of sign by clicking on the appropriate toolbar button  ,  or  then click onto the video at the position the sign should be placed. The only sign that requires further editing is the speed limit sign. When selected, a speed limit sign displays two buttons on the right side of the icon, one with a plus sign, the other with a minus sign. These buttons can be clicked on to adjust the speed limit associated with the sign with a step of 5 km/h. Figure 2.5.1.3.b. shows two signs placed on a road segment, the speed limit sign is selected.

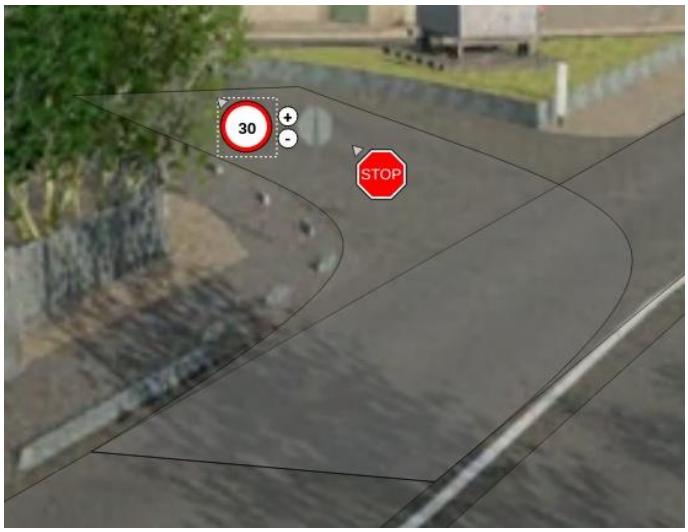


Figure 2.5.1.3.b - Traffic sign editor in the RES application

2.5.5. Level crossing zone editor

The level crossing zone editor allows the RES user to specify the location of the level crossing in the scene frame with respect to the road infrastructure. The application only supports a single level crossing zone per project, i.e., one video. Therefore, if the user attempts to create a new zone while one already exists, the old zone is automatically removed. To create a level crossing zone, the RES

user must first click on the corresponding toolbar button  then click on the video to place the points that define the polygon of the zone. The edition of the zone is terminated when the RES user clicks on the first point of the zone in order to close the polygon. Figure 2.5.1.4. shows the level crossing zone editor in the RES application.

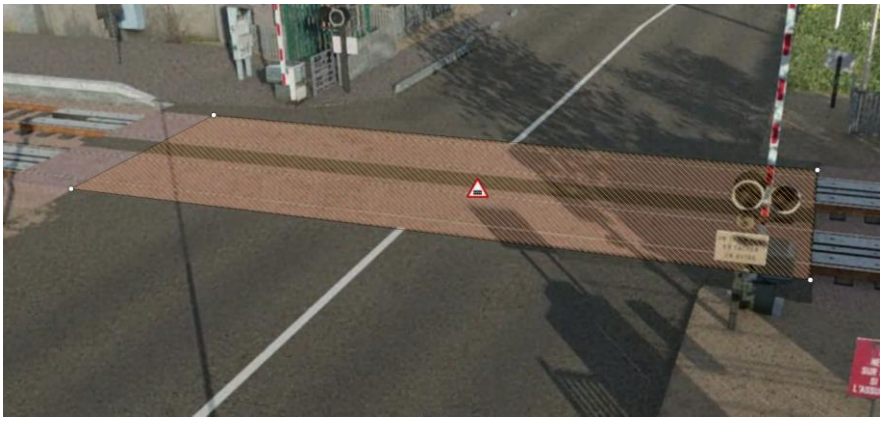


Figure 2.5.1.4 - Level crossing zone editor in the RES application

2.5.6. New project dialog

The first step in using the RES is to create a new project. A project is a file that contains all the data input by the RES user, the parameters and output of each stage. A project is associated to a single video file. To create a new project, the RES user must click on the corresponding toolbar button or select the menu item “File/New” and fill out a form as shown in figure 2.5.2. The new project dialog asks the RES user to specify the input video file as well as the camera’s intrinsic and extrinsic parameters needed to undistort images and project position from the camera reference frame to the scene reference frame.

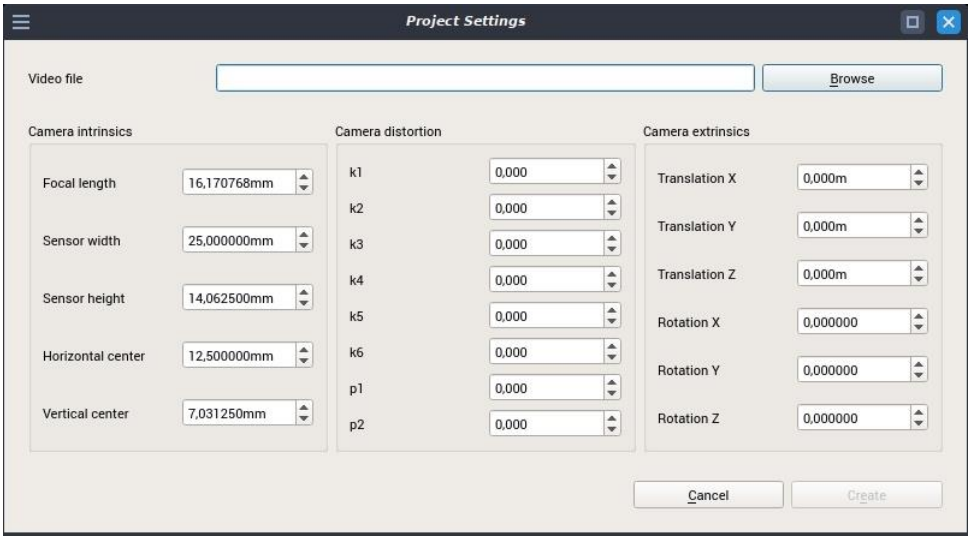


Figure 2.5.2 - The project dialog of the RES application

2.5.7. Detection stage dialog

The detection stage dialog is displayed when the RES user clicks on the corresponding toolbar button  . The RES user can modify the parameters of all three parts of the detection stage as shown in figure 2.5.3.

For the object detection part, the user can select the three files related to the YOLO model namely the HDF5 file containing the weights of the network, the file defining the default anchors and the file containing the class names. If the hardware that executes the RES has multiple GPUs, it is possible to specify the number of GPU to dedicate to the object detection. The score threshold defines the minimum confidence the model must have for a detection to be considered correct and be part of the output. The IOU (intersection over union) threshold can be adjusted. This parameter defines the ratio of areas for two bounding boxes to be merged. This process is called non-maximum suppression and is a common post-process for object detection algorithms.

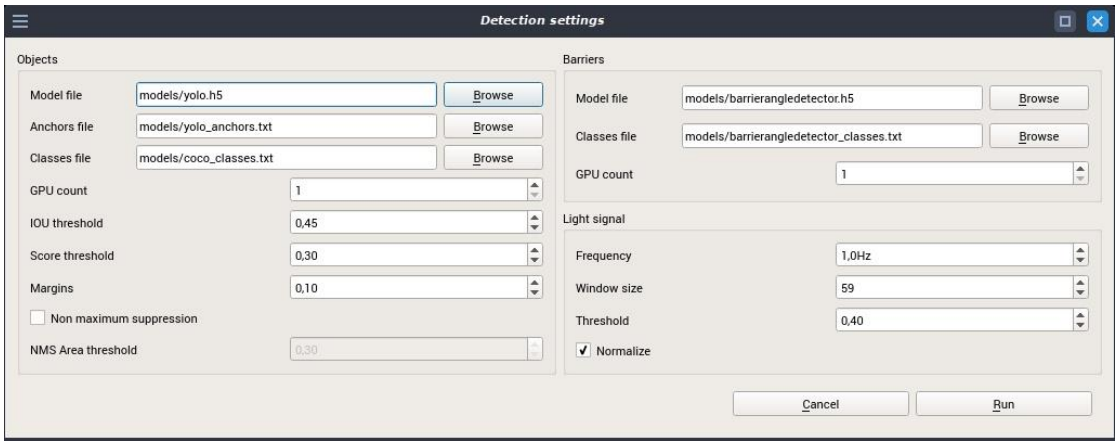


Figure 2.5.3 - The detection stage settings dialog of the RES application

Similarly, to the object detection part, the barrier detection part requires the RES user to input the weight file and the file containing the class names. It is also possible to execute the model on multiple GPUs in order to improve the computation speed.

The light signal detection part requires the RES user to specify the frequency of the signal if it is blinking. A frequency of 0Hz indicates the signal is constant. The RES user must also specify the size of the time window used to apply he Goertzel algorithm on the signal as well as the strength threshold that triggers the detection. Finally, a check box is provided to normalize the input signal before applying the Goertzel algorithm as explained in section 2.2.12.2.1.

2.5.8. Tracking stage dialog

The tracking stage dialog, shown in figure 2.5.4, is displayed when the RES user clicks on the corresponding toolbar button  .

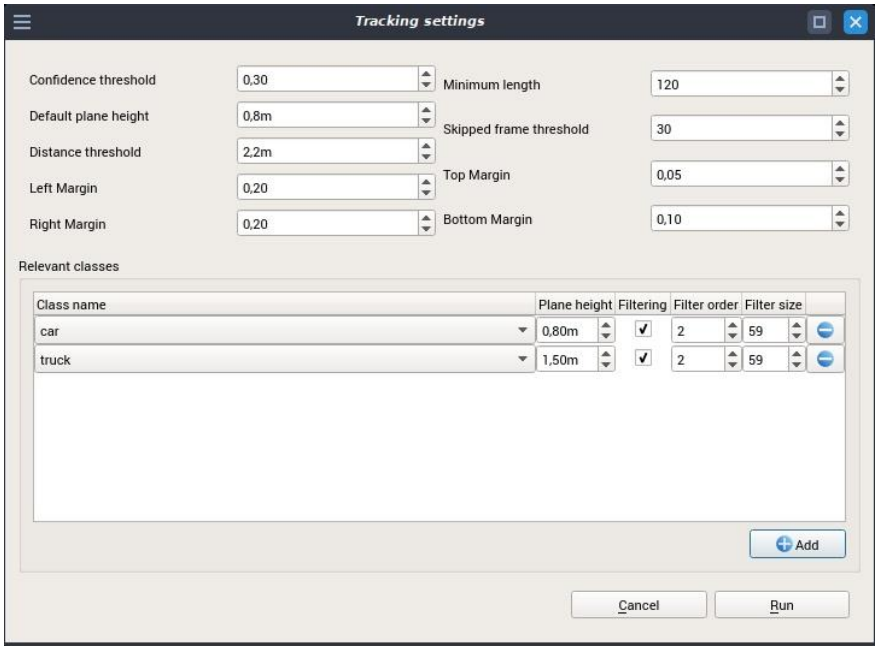


Figure 2.5.4 - The tracking stage settings dialog of the RES application

Only the relevant classes selected by the RES user will be tracked. The central part of the dialog contains the parameters specific to each relevant class like the height of the virtual plane in which the object centroids are, the low-pass filter parameters. The class selection field only displays classes for which at least one detection exists in the project.

The confidence threshold is similar to the one of the detection stages. It allows the RES user to limit the observations to those whose score is higher than the threshold. This allows restricting the number of observations without the need to redo the detection stage.

The default plane height is used to project the centroid of a bounding box associated to a non-relevant class. Since an object can be misclassified, non-relevant tracks are only removed at the end of the process.

The distance threshold controls the maximum distance the predicted position of the centroid must be distant from an observation for the association to be valid.

The minimum length defines the minimum number of consecutive frames a track must have to be considered valid and the skipped frame threshold controls the maximum number of consecutive frames in which a tracked object must not be detected for the track to be stopped.

The margin parameters allow to specify the portion of the video in which the centroid of a bounding box must be for the detection to be considered correct: when an object enters the field of view of the camera, its bounding box is sometimes incomplete since most of the object is still not visible by the camera and, therefore, the centroid of the box will not match the projection of the object's centroid in the camera reference frame. The detection margin parameters are relative to the image size.

2.5.9. Recognition stage dialog

The recognition stage dialog, shown in figure 2.5.5, is displayed when the RES user clicks on the corresponding toolbar button  .

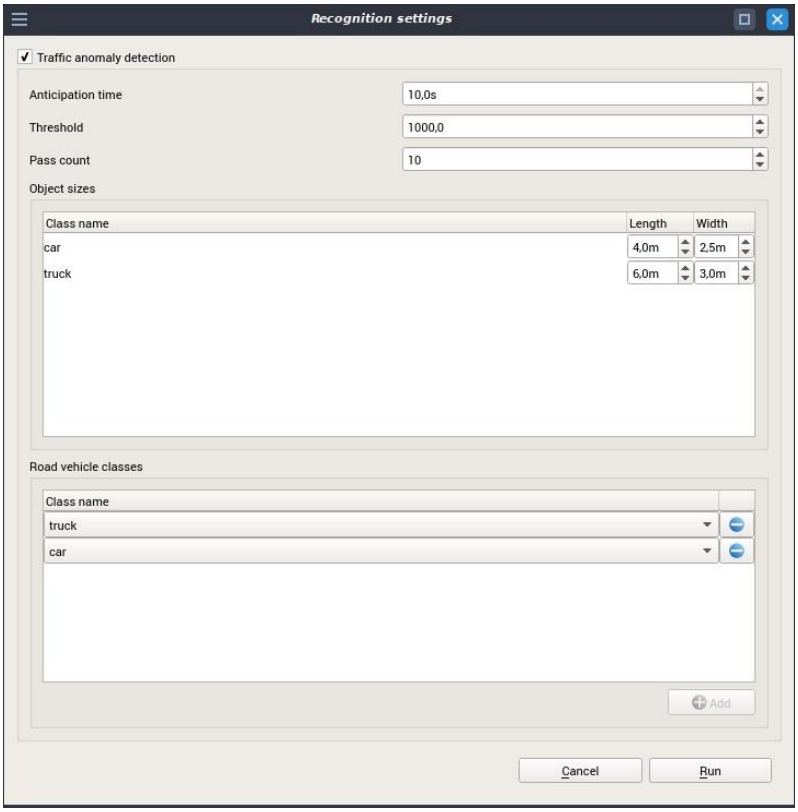


Figure 2.5.5 - The recognition stage settings dialog of the RES application

Activity recognition has no parameter and the dialog only displays the parameters of the traffic anomaly detection.

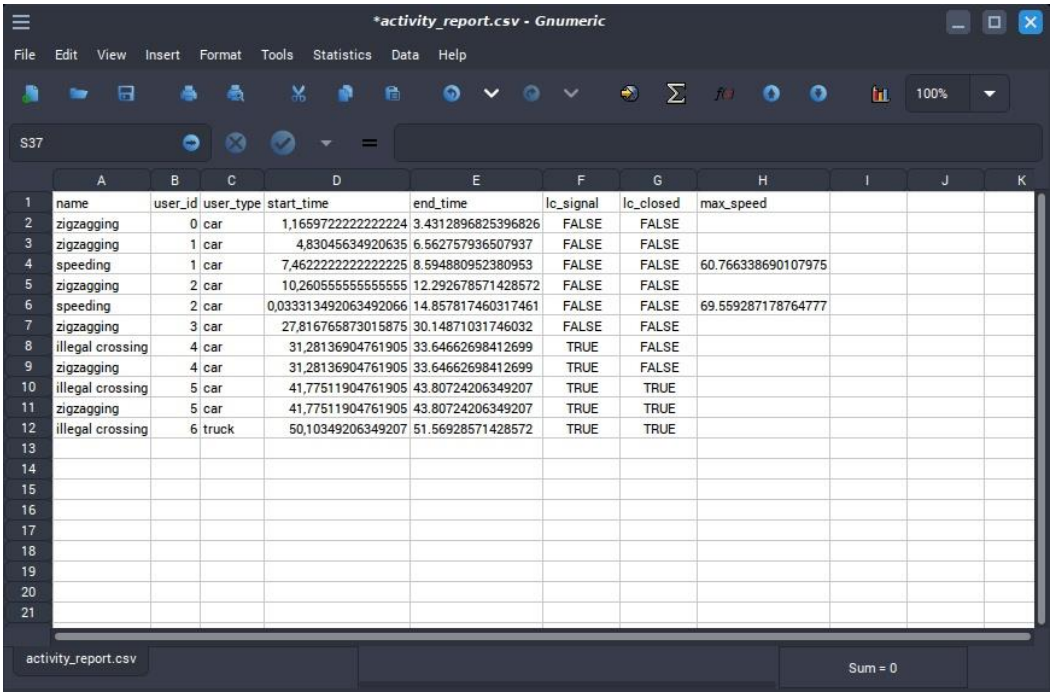
Anomaly detection requires the RES user to select the classes that correspond to road vehicles and provide their average dimensions. The RES user can also adjust the number of times a road vehicle will be simulated and the threshold distance between the nearest reference normal trajectory and the actual one that will trigger the anomaly detection.

The anticipation time parameter, controls the time duration an animated entity is put in the simulation before its trajectory actually starts as explained in section 2.4.1.

2.5.10. Activity report

Once all three stages of the RES have been executed on a video, the RES user can export an activity report in CSV format by clicking on the toolbar button with the following icon  .

The activity report file can be opened in a spreadsheet software in order to be further analyzed as shown in figure 2.5.6.



	A	B	C	D	E	F	G	H	I	J	K
1	name	user_id	user_type	start_time	end_time	lc_signal	lc_closed	max_speed			
2	zigzagging	0	car	1,1659722222222224	3.4312896825396826	FALSE	FALSE				
3	zigzagging	1	car	4,83045634920635	6.562757936507937	FALSE	FALSE				
4	speeding	1	car	7,4622222222222225	8.594880952380953	FALSE	FALSE	60.766338690107975			
5	zigzagging	2	car	10,2605555555555555	12.292678571428572	FALSE	FALSE				
6	speeding	2	car	0,033313492063492066	14.857817460317461	FALSE	FALSE	69.559287178764777			
7	zigzagging	3	car	27,816765873015875	30.14871031746032	FALSE	FALSE				
8	illegal crossing	4	car	31,28136904761905	33.64662698412699	TRUE	FALSE				
9	zigzagging	4	car	31,28136904761905	33.64662698412699	TRUE	FALSE				
10	illegal crossing	5	car	41,77511904761905	43.80724206349207	TRUE	TRUE				
11	zigzagging	5	car	41,77511904761905	43.80724206349207	TRUE	TRUE				
12	illegal crossing	6	truck	50,10349206349207	51.56928571428572	TRUE	TRUE				
13											
14											
15											
16											
17											
18											
19											
20											
21											

Figure 2.5.6 - An activity report generated by the RES application in a spreadsheet software.

The activity report contains a list of all activities detected by the RES. Each activity has the following attributes: the unique identifier of the LC user that is the source of the activity, its type, the time in seconds from the beginning of the video at which the activity starts and ends and the status of the level crossing at the beginning of the activity, i.e., whether the light signals were active or not and whether the barriers were opened or closed.

Some activities may have additional attributes, for example, the speeding activity also has an attribute that contains the maximum recorded speed of the LC user.

3. EXPERIMENTATIONS AND RESULTS

The RES was developed with the Python 3.5 programming language, deep learning models are executed using the Keras[5] framework and the graphical interface is based on the PyQt5 API (a Python wrapper for the C++ Qt5[6] API). All experiments were realized on a Asus ROG Zephyrus S GX535 Laptop with an NVidia RTX 2080 MaxQ GPU, an Intel(R) Core(TM) i7-8750H CPU (up to 4.1 GHz) and 24 Gb of RAM. The barrier angle detector was trained on a Dell Aurora R7 PC with two NVidia GeForce GTX 1080 Ti, an Intel(R) Core(TM) i7K-8700K CPU (up to 4.6 GHz) and 32 Gb of RAM.

3.1. Synthetic video generation

In order to evaluate the performance of the RES, we generated a set of videos using our simulator. These videos were generated by recording a simulated level crossing in a virtual 3D environment, in which a dense traffic crosses a level crossing on a standard 1 by 1 road, i.e., two opposite lanes. In order to not introduce a bias in this study, we used different level crossing models and environments than those used to generate the dataset for training the barrier angle detector.
In total, 16 videos were generated each with a 24 minutes duration. Each time the level crossing was simulated, two videos were recorded simultaneously from two cameras with the same intrinsic parameters that cover both sides of the level crossing, with an overlap centered on the level crossing itself.

During the 24 minutes, the time controlling the lighting and weather system in the simulator is accelerated by a factor of 30, making the weather and lighting evolve as if 12 hours had passed. The vehicle dynamics and behaviors of the simulated driver are unaffected by this time scale. The weather evolves from partly cloudy at the beginning of the video, rainy with fog at the middle and ends with a clear sky but with the environment still wet. We did not test the RES with snow. Though it is possible to record a video with the snow falling, accumulating and melting in our simulator. However, we do not currently have a model to simulate the tracks of the vehicle wheels in the snow and we judged the videos would not look real enough without it (this could potentially affect the object detection negatively).

The traffic is composed of one type of truck and three types of cars. The paint of all the vehicles is randomized in order to provide more variants. The traffic generator is set to generate 5% of trucks and 95% of cars.

During the recording of the videos, the groundtruth is also recorded using data from the simulation. For each video frame, this groundtruth contains the bounding boxes in the camera reference frame of all visible vehicles as well as the position and velocity of their centroids in scene reference frame.

[5] https://keras.io/
[6] https://www.qt.io/

It also contains the angle of each barrier in the camera reference frame and the state of light signals. However, the groundtruth does not take into account occlusions, i.e., vehicle data is recorded in the ground truth even if the vehicle is not actually visible in the video because it is hidden behind a truck. Because of this, the number of false positives presented in this document is actually higher than what it should be, and thus the reported accuracies are actually lower than what they would be if the ground truth was manually generated by a human operator.

3.2. Detection stage performance evaluation

Although the RES is an offline process, the analysis of a video should be completed in a reasonable time. The most time-consuming tasks of the process are the object and barrier angle detection phases. One way to reduce the overall computational complexity of the analysis is to reduce the input video resolution. However, reducing the input video resolution implies a loss of information that may also reduce the ability of the system to properly detect objects, track them and then recognize dangerous behaviors.

In this section we present the overall performance of the detection stage while investigating the impact of reducing video resolution on the components of the RES, namely the object and barrier detectors. As we will show in the rest of this document, performance of the tracking and activity recognition is highly dependent on the performance of the detectors. The performance of the light signal detector is unaffected by the resolution and is therefore excluded from this experiment.
In order to study the impact of reducing video resolution on the system, we used two 24 minutes length videos, recording the same exact scene from both sides of the level crossing, with a resolution of 1280x720 pixels (720p) at 30 frames per second.

The two videos were resized using Lanczos interpolation to match the following resolutions: 854x480 (480p), 640x360 (360p) and 426x240 (240p).

The detection stage was then executed on each video using the same exact parameter set.

3.2.1. Object detector performance

In order to evaluate the performance of the object detection on each video, we used the current PASCAL VOC Challenge metrics (Everingham et al. 2010), a very popular set of measures used to evaluate and compare object detection algorithms. We used the most recent implementation7 of the metrics, which interpolates all data points instead of the 11-point interpolation approach described in the paper. The metrics are computed for each of the 43200 frames of the input videos.
The total number of true positives (TP), false positives (FP), false negatives (FN) and average precision (AP in percent) for each class is shown in table 3.2.1.a.

[7] https://github.com/rafaelpadilla/Object-Detection-Metrics

		Car				Truck			
Camera	**Resolution**	**TP**	**FP**	**FN**	**AP**	**TP**	**FP**	**FN**	**AP**
North	720p	81092	5802	2596	93	1076	3195	1599	27
South	720p	80335	4015	2686	94	1900	5441	848	54
North	480p	81521	5309	2167	95	1612	10375	1063	25
South	480p	81082	4483	1939	97	2017	6636	731	59
North	360p	80948	9456	2740	91	1799	12496	876	27
South	360p	80518	8835	2503	94	2135	6549	613	63
North	240p	73890	6837	9798	85	1951	12763	724	40
South	240p	77899	5555	5122	93	2033	6325	715	59

Table 3.2.1.a - Performance of object detection for different video resolutions

The precision/recall curves are shown in figure 3.2.1.

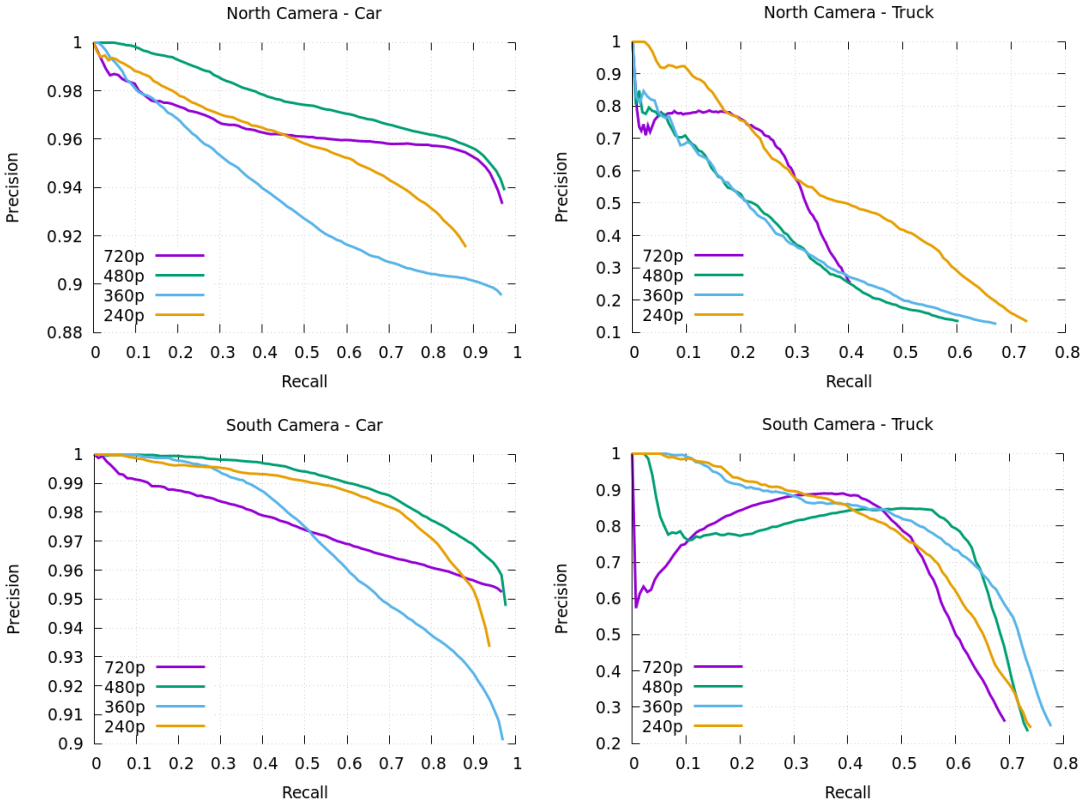


Figure 3.2.1 - Precision/Recall curve for each class

Table 3.2.1.b shows the timing, detection speed in frames per second (FPS) and mean average precision (mAP) for each test video.

Camera	Resolution	Timing	FPS	mAP
North	720p	2h20m	5	60%
South	720p	2h20m	5	74%
North	480p	1h41m	7	60%
South	480p	1h43m	7	78%
North	360p	1h23m	8	59%
South	360p	1h22m	8	79%
North	240p	1h04m	11	62%
South	240p	1h05m	11	76%

Table 3.2.1.b - Timing results for different video resolutions

From these results, we can see that the overall precision of object detection remains stable across the video resolution that we tested and is on par with the reported performances[8] of YOLO on the COCO dataset, although we only detect two classes of objects in our study. However, per-class precision varies differently.

Car detection is good at any resolution except for the 240p resolution for which the average precision decreases significantly. We can also observe that the number of FP at the 360p resolution almost doubles compared to the higher resolutions, this could be a problem for object tracking as it may result in many invalid tracks or poor accuracy due to shifting bounding boxes centroids. For this class (car), the detector seems to perform better at the 480p resolution.

Truck detection is not very good at any resolution. Average precision is highest at the 240p but the number of FP is also very high and would negatively impact tracking. One possible explanation for this low performance could be that the 3D model of truck that we used in our simulation is very much different from the images of trucks present in the training dataset. For this class (truck), the detector seems to perform slightly better at the 240p resolution.

3.2.2. Barrier detector performance

To evaluate the performance of the barrier angle detector, we chose simple metrics. Since we approached the barrier detection from a classification perspective, we calculated the categorical accuracy (CA), i.e., the percentage of frames in which the detector got the angle right. Since the

[8] https://pjreddie.com/darknet/yolo/ (see Performance on the COCO Dataset)

classification could be wrong but the predicted angle could be close to the actual one, we also calculated the mean (μ), maximum absolute value (max) and the standard deviation (σ) of the angular error (in degrees). These three values give us an idea of the distribution of the error.
Similarly, to the object detection performance, we also investigated the impact of video resolution on the error.

In this test, we compared the raw output of the detector to the ground truth, i.e., we did not apply the disambiguation rules nor the angle tracking described in section 2.2.2. Table 3.2.2.a shows the results on our test videos for each of the two barriers of the level crossing.

		North barrier				South barrier			
Camera	**Resolution**	**CA**	**μ**	**σ**	**max**	**CA**	**μ**	**σ**	**max**
North	720p	85%	1.6	17.5	177	89%	0.4	7.1	171
South	720p	93%	-1	14.9	177	63%	-3.2	15.4	171
North	480p	86%	0.3	15	177	88%	1.85	15	174
South	480p	90%	1.6	16.5	177	77%	-0.8	5.2	165
North	360p	82%	-0.4	16	177	85%	3.7	19.8	174
South	360p	82%	2.4	20.2	177	76%	-0.82	5.5	171
North	240p	78%	0	23.2	177	73%	8	27.5	174
South	240p	54%	-0.9	29	177	83%	-1.3	9.8	162

Table 3.2.2.a - Performance of barrier angle detection for different video resolutions

The results of this test suggest that the video resolution has little to no effect on the performance of the barrier angle detection, except for the lowest resolution on the video of the south camera.

We also evaluated the performance of the barrier detector with filtering and disambiguation. The metrics are the same except that we removed the categorical accuracy. Indeed, since the barrier is tracked, the filtered angle is a continuous value and not a category.

Results are displayed in table 3.2.2.b.

		North barrier			South barrier		
Camera	**Resolution**	**μ**	**σ**	**max**	**μ**	**σ**	**max**
North	720p	-0.5	13.6	147	0.2	4.1	107
South	720p	1.4	9.6	110	-2	9.8	130
North	480p	-1.2	10.1	145	0.8	9.2	131
South	480p	1.2	7.5	84	-0.6	2.4	75
North	360p	-1.6	12.6	119	2.8	14.8	141
South	360p	2.69	13.0	102	-0.7	3.2	80
North	240p	-2.5	17.5	125	5.6	19.5	137
South	240p	4.7	27.1	154	-0.8	5.1	91

Table 3.2.2.b - Performance of barrier angle detection for different video resolutions with filtering and disambiguation

We can see that our filtering and disambiguation process reduces the standard deviation of the error as well as its maximum absolute value. However, significant errors still occur especially when most of the barrier is occluded by a large vehicle (like a truck). We expect this problem to be significantly reduced by retraining the model with a dataset that contains a greater proportion of images with partly occluded barriers.

3.2.3. Light signal detector performance

In order to evaluate the performance of the light detector, we calculated the Pearson correlation coefficient between the detected signal and the ground truth. This coefficient is computed using the following equation:

$$r = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}}$$

where x_i and y_i are the amplitudes, respectively, of the detected and ground truth signals samples i, and $\bar{x}$ and $\bar{y}$ are the mean amplitudes.

Results are displayed in table 3.2.4 for both cameras of the test video.

Camera	Correlation
North	0.98
South	0.98

Table 3.2.4 - Pearson's correlation coefficients between detected light signal state and ground truth

The detected signal, ground truth and error for the north camera are shown in Figure 3.2.4. Results for the south camera are similar. The size of the window for Goertzel algorithm was set to 89 samples and the detection threshold to 0.3.

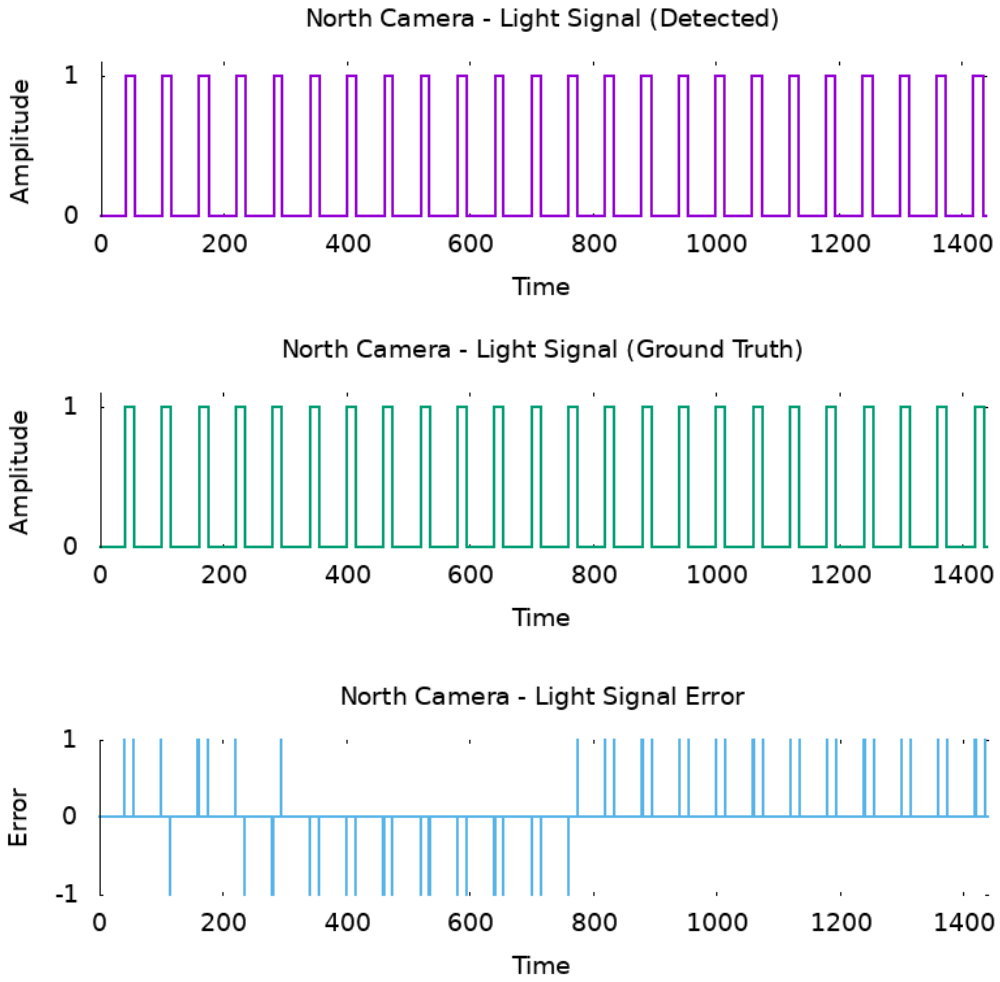


Figure 3.2.4 - Light signal state detected, ground truth and error

We can see that the detector performs very well for the whole 24 minutes of the test video on both cameras. The errors only occur when the state of the light signal changes and are due to the Goertzel algorithm that works with a window. Indeed, when the algorithm detects an oscillation, it is not possible to determine precisely at which sample, in the time window, it starts or ends. We chose the middle of the window as reference. The error could be reduced by decreasing the window size but at the price of higher risks of false negatives. The detector performance is unaffected by the resolution of the input video as long as the region of interest of each light signal is non empty.

3.3. Tracking stage performance evaluation

We evaluated the performance of the tracking stage using the py_motmetrics[9] Python library. The library computes the CLEAR MOT metrics (Bernardin and Stiefelhagen 2008), namely the multiple object tracking precision (MOTP) and multiple object tracking accuracy (MOTA).

MOTA combines three error sources: false positives, missed targets and identity switches, while MOTP measures misalignment between ground truth and detected tracks.

The library also computes the mostly tracked (MT) measure, i.e., the number of ground-truth trajectories that are covered by a track hypothesis for at least 80% of their respective life span, the mostly lost (ML) measure, i.e., number of ground-truth trajectories that are covered by a track hypothesis for at most 20% of their respective life span, the partially tracked (PT) measure i.e. the number of objects tracked between 20 and 80 percent of lifespan, the fragmentation (FM), i.e., the total number of switches from tracked to not tracked and the number of ID switches (IDs) (Ristani et al. 2016).

Global track quality measures are also provided, including the recall (Rcll), i.e., the number of detections over number of objects, precision (Prcn), i.e., the number of detected objects over the sum of detected and false positives (FP). We also included the total number of false negatives (FN). Tracking performance is evaluated on the same two videos used to evaluate the performance of the detection stage as well as the rescaled videos. The number of tracks in the ground truth is 480. Results are shown in table 3.3, ↑ indicates that the higher is better and inversely ↓ indicates that the lower is better.

Camera	North	South	North	South	North	South	North	South
Resolution	720p	720p	480p	480p	360p	360p	240p	240p
Rcll ↑	88%	85%	87%	85%	85%	80%	75%	75%
Prcn ↑	87%	88%	83%	86%	81%	82%	78%	82%
MT ↑	339	342	337	320	339	278	251	263
PT ↓	116	103	117	128	103	137	160	130
ML ↓	25	35	26	32	38	65	69	87
FP ↓	10901	9902	15620	12498	16950	14803	18055	14369
FN ↓	11444	12933	11372	12926	12617	17346	21378	21314
FM ↓	224	226	191	216	174	198	202	269

[9] https://github.com/cheind/py-motmetrics

IDs ↓	47	42	85	38	66	53	71	43
MOTA ↑	74%	73%	69%	70%	66%	63%	54%	58%
MOTP ↓	1.34	1.35	1.19	1.25	1.16	1.28	1.22	1.37

Table 3.3. - Tracking performance

We can see that the performance of our tracker is moderately good. Since our approach for the tracking algorithm is not as sophisticated as the state of the art, its performance is directly related to the quality of the detection. Also, since detection of trucks is poor with the combination of the pre-trained version of YOLO and our test videos, almost every truck is either misclassified (this implies that the trajectory is not projected at the correct height) or barely tracked at all. Although using a state of the art tracker would almost certainly improve the precision of object tracking (i.e., reduce the distance error between ground truth and detected tracks), it is likely that the most important factor to improve the tracking performance is the quality of the object detection.

3.4. Recognition stage performance evaluation

In order to evaluate the performance of the recognition stage, we applied our activity detectors on the ground truth tracks for each test video. The information about the position of the level crossing and the road infrastructure, the position of the barriers and light signal states are therefore the same for both our ground truth activities and the detected ones.

We executed the detection and tracking stages on 14 videos at the 480p resolution before running the recognition stage. These 14 videos show 7 scenes from both the north and south cameras. In each scene, every vehicle performs the same exact activity (illegal crossing, queuing, stopping, wrong-way crossing, zigzagging). Since wrong-way crossing and zig-zagging can only be performed when the traffic is going in one single direction, we generated two separate scenes for both directions. Performance of the first two stages on these test videos can be found in Annex A (object detection) and Annex B (object tracking).

For the speeding activity, we used the same video as in sections 3.2 and 3.33.3 at the 480p resolution. We only changed the speed limit signs present in the project to 20 km/h instead of 50 km/h.

We used a methodology similar to that used for regular object detection performance evaluation: we attempted to match each activity in the ground truth to a detected one. A ground truth activity and a detected one are matched if they belong to the same category and if the ratio of intersection over union (IOU) of their frame intervals (the interval defined by the first and last video frame in which the activity is detected) is greater than 0.5. If a match exists, the number of true positives (TP) is incremented and the matched activities are removed from the set so as to not count them more than once. If no match exists, the number of false negatives (FN) is incremented. At the end of the

process, if there are still some unmatched activities in the detected set, they are counted as false positives (FP). We then calculate the precision and recall for each video using the following formulas:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Results are displayed in tables 3.4, ↑ indicates that the higher is better and inversely ↓ indicates that the lower is better.

Video	Camera	TP ↑	FP ↓	FN ↓	Precision ↑	Recall ↑
illegal crossing	North	431	3	42	99%	91%
illegal crossing	South	447	5	30	99%	94%
queuing	North	86	24	72	78%	54%
queuing	South	86	14	71	86%	55%
stopped	North	126	9	62	93%	67%
stopped	South	144	5	48	97%	75%
wrong way (North)	North	211	19	15	92%	93%
wrong way (North)	South	212	4	14	98%	94%
wrong way (South)	North	253	3	15	99%	94%
wrong way (South)	South	247	0	19	100%	93%
zig-zagging (North)	North	187	17	45	92%	81%
zig-zagging (North)	South	216	1	12	100%	95%
zig-zagging (South)	North	204	6	41	97%	83%
zig-zagging (South)	South	223	3	20	99%	92%
speeding[10]	North	401	156	123	72%	77%
speeding[11]	South	358	105	119	77%	75%

Table 3.4 - Activity recognition performance

[10] The test video contains 415 instances of speeding and 109 instances of illegal crossing.
[11] The test video contains 416 instances of speeding and 61 instances of illegal crossing.

Activity recognition performance is quite good despite the low quality of truck detection and tracking (which account for most of the false negatives). False positives are mostly caused by objects that are detected multiple times, often with a different class in the same frame. This problem could be alleviated with the retraining of the YOLO detector.

Speeding activity is quite challenging for our detector mainly because the speed of an object is derived from its position across time. Even with our low-pass filter, the position of an object centroid often jitters from frame to frame, which adds a lot of noise to the speed estimation.

4. CONCLUSIONS

In this document, we presented the Risk Evaluation System, an offline process that has been developed by UTBM in collaboration with CEREMA of Toulouse, for a comprehensive evaluation of the dangerous situations and, thus, human risk, by analyzing a large number of consecutive videos recorded at a LC and extracting behaviors.

A trajectory-based approach to behavior recognition was adopted and the system was decomposed in three sequential stages: detection stage, tracking stage and recognition stage.

The detection stage uses deep-learning models (YOLO and ResNet50) to detect LC users and the angle of the barriers, and it uses a signal processing approach to extract the state of the light signals. The tracking stage uses a custom-made scene-space tracker that exploits the fact that the recording camera is fixed, and its pose known.

The recognition stage uses the trajectories generated by the tracking stage together with information about the road and LC infrastructure in order to detect several dangerous activities (illegal crossing, queuing, stopping, wrong-way crossing, zig-zagging, speeding and abnormal crossing).
Performance of the system was evaluated on a set of synthetic videos that were generated from an advanced simulator that combines lighting and weather simulation, realistic vehicle dynamics and artificial intelligence-based driver behavior.

Future works include evaluating the performances of the system on video recording of real LCs, improving the object detection performance (especially for trucks) by retraining YOLO with a more complete training set, improving the tracking performance by including objects appearances in data assignment cost measures, and validating our simulation-based traffic anomaly detector.

Additionally, level crossing state detection could be improved. Although our barrier angle detector can still estimate the angle of a broken barrier as long as the red and white pattern is still visible on the remaining portion, the system could benefit from an actual detection of a barrier being broken. This event could be recorded in the database as a dangerous behavior. On some level crossings, malfunction of the light signals may change the frequency of blinking lights. Our light signal detector would fail in those cases without an intervention of the operator to change the reference frequency. A more robust detector that could also detect and record the malfunction would be very useful.

Use cases for pilot tests should focus on the current set of behavior the system can detect. However, other types of dangerous behaviors could be simulated in a controlled LC environment to validate the anomaly detector.

5. REFERENCES

Bernardin, Keni, and Rainer Stiefelhagen. 2008. “Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics.” *EURASIP Journal on Image and Video Processing*.

Buisson, Jocelyn, Nicolas Gaud, Stéphane Galland, Mikaël Gonçalves, and Abderrafiaa Koukam. n.d. “Toward an Environment for the Simulation of Heterogeneous Entities in Virtual Cities.” In *International Workshop on Environments for Multiagent Systems (E4MAS14)*. IFAAMAS.

Carnegie-Mellon University. Robotics Institute, and R. Craig Coulter. 1992. *Implementation of the Pure Pursuit Path Tracking Algorithm*.

Elek, Oskar, and Petr Kmoch. 2010. “Real-Time Spectral Scattering in Large-Scale Natural Participating Media.” *Proceedings of the 26th Spring Conference on Computer Graphics - SCCG ’10*.

Everingham, Mark, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. 2010. “The Pascal Visual Object Classes (VOC) Challenge.” *International Journal of Computer Vision*.

Gipps, P. G. 1981. “A Behavioural Car-Following Model for Computer Simulation.” *Transportation Research Part B: Methodological*.

Goertzel, Gerald. 1958. “An Algorithm for the Evaluation of Finite Trigonometric Series.” *The American Mathematical Monthly*.

Guttman, Antonin, and Michael Stonebraker. 1983. *R-Trees: A Dynamic Index Structure for Spatial Searching*.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. “Deep Residual Learning for Image Recognition.” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Kalman, R. E. 1960. “A New Approach to Linear Filtering and Prediction Problems.” *Journal of Basic Engineering*.

Meeus, Jean. 1998. *Astronomical Algorithms*.

Perronnet, Florent. 2015. “Régulation Coopérative Des Intersections: Protocoles et Politiques.” PhD, UTBM.

Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. “You Only Look Once: Unified, Real-Time Object Detection.” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ristani, Ergys, Francesco Solera, Roger Zou, Rita Cucchiara, and Carlo Tomasi. 2016. “Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking.” *Lecture Notes in Computer Science*.

Savitzky, Abraham, and M. J. E. Golay. 1964. “Smoothing and Differentiation of Data by Simplified Least Squares Procedures.” *Analytical Chemistry*.

Treiber, M., A. Hennecke, and D. Helbing. 2000. “Congested Traffic States in Empirical Observations and Microscopic Simulations.” *Physical Review. E, Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics* 62 (2 Pt A): 1805–24.

Zheng, Binqi, Pengcheng Fu, Baoqing Li, and Xiaobing Yuan. 2018. “A Robust Adaptive Unscented Kalman Filter for Nonlinear Estimation with Uncertain Noise Covariance.” *Sensors* 18 (3).

ANNEXES

A. Performance of the object detection on the activity detection test videos

		Car				Truck			
Video	**Camera**	**TP**	**FP**	**FN**	**AP**	**TP**	**FP**	**FN**	**AP**
illegal crossing	North	42298	3333	918	96	512	5614	316	29
illegal crossing	South	42426	2522	543	98	557	5081	266	49
queuing	North	265069	21502	21482	90	6330	19895	1592	55
queuing	South	253114	13018	16175	93	6377	15451	2569	61
stopping	North	185360	10191	5089	96	2981	15901	435	61
stopping	South	208697	7182	5516	97	1680	6531	1607	44
wrong way (North)	North	30280	1864	1201	93	28	6956	878	0
wrong way (North)	South	29993	1560	301	97	735	3028	161	72
wrong way (South)	North	37483	2821	48	98	691	2798	102	78
wrong way (South)	South	30803	2278	55	99	223	932	542	16
zigzagging (North)	North	59203	4570	412	98	612	4287	1404	9
zigzagging (North)	South	72032	3116	261	99	928	3782	1102	32
zigzagging (South)	North	80122	3971	2702	95	1403	9224	831	50
zigzagging (South)	South	64883	1611	4253	93	232	4637	1917	1

B. Performance of the tracking on the activity detection test videos

Video	illegal crossing	illegal crossing	queuing	queuing	stopping	stopping
Camera	North	South	North	South	North	South
Rcll ↑	80%	81%	85%	89%	94%	93%
Prcn ↑	84%	85%	78%	87%	86%	90%
MT ↑	302	322	143	153	186	178
PT ↓	140	128	55	45	7	13
ML ↓	35	27	1	2	1	2
FP ↓	6584	6418	72367	37208	30416	21982
FN ↓	8800	8467	43179	31936	12168	14346
FM ↓	173	306	276	198	95	129
IDs ↓	3	1	295	240	195	131
MOTA ↑	65%	66%	61%	75%	78%	83%
MOTP ↓	1.24	1.32	1.22	1.24	0.92	1

Video	wrong way (North)	wrong way (North)	wrong way (South)	wrong way (South)
Camera	North	South	North	South
Rcll ↑	88%	88%	88%	82%
Prcn ↑	80%	90%	87%	88%
MT ↑	161	154	200	188
PT ↓	43	57	53	63
ML ↓	9	2	6	7
FP ↓	7122	2963	5109	3407
FN ↓	3848	3895	4701	5858

FM ↓	93	98	69	106
IDs ↓	21	10	15	0
MOTA ↑	66%	78%	74%	71%
MOTP ↓	1.3	0.99	1.08	1.33

Video	**zigzagging (North)**	**zigzagging (North)**	**zigzagging (South)**	**zigzagging (South)**
Camera	North	South	North	South
Rcll ↑	86%	90%	96%	87%
Prcn ↑	85%	90%	81%	86%
MT ↑	173	191	216	167
PT ↓	49	32	19	59
ML ↓	2	1	0	8
FP ↓	9448	7771	19362	9771
FN ↓	8811	7453	3742	9566
FM ↓	85	94	130	134
IDs ↓	73	55	120	66
MOTA ↑	70%	79%	73%	73%
MOTP ↓	1.05	1.01	1.13	1.23