SEVENTH FRAMEWORK PROGRAMME

VIT
Vision for Innovative Transport

Project partly funded by the EC
Grant agreement no. *222199*

SP4-Capacities - Research for SMEs

REPORT ON SOFTWARE PACKAGE OF TRAIN PROFILING OCV AND TRAIN LOAD VERIFICATION

Deliverable D4.2

Release date 15 January 2010

Work package number WP4
Work package title 2D Train Scanning
Activity Type RTD

About the Document

This document reports the technical details of the software package developed within WP4 on 2D train profile construction and container code verification. It also describes the final field tests carried out at Vado Ligure (IT) plant showing the effectiveness of the developed modules.

This document, together with the software release related to it, is *Project Deliverable D4.2.*

The document has been produced by the collaboration of the workpackage WP4, the participants of the workpackage have all duly contributed to the activity of the workpackage and the production of this document and they endorse the final version as the conclusion of the workpackage.

Workpackage leader

Francesca Odone (DISI)

Document authors

Francesca Odone (DISI)
Alberto Lovato (IMA)
Emanuele Trucco (DUN)

Document reviewers

Renzo Ferraris (ILOG)
Thomas Keese (WIT)
Michele Molinari (MOL)

Table of contents

INTRODUCTION

This report describes the last 6 months of RTD activity carried out within WP4. In particular, it illustrates the final version of the software modules performing the 2D reconstruction of the train profile and the optical character verification of container codes. It summarizes the implemented solutions, highlighting the changes with respect to previous deliverables due to the results obtained during field tests. Also it describes in details the experimental analysis carried out on the field, at months 16 to 18 of the project.

Structure of the report

The report is organized with the following structure:

1. Overview
2. Reconstruction of the train profile and automatic load verification (including experiments)
3. Localization and recognition of the ownership code (including experiments)
4. Description of the software package (including examples of use)
5. Related bibliography

AUDIENCE

The present deliverable is filed as Confidential, as it contains critical information for the VIT project and also for the Metrocargo system.

Therefore the audience of the document is restricted the project participants --- the SME’s who will find the technical details following their user requirements and the RTD performers who will use the present report as a guideline of their research and development activity.

1. Overview

This section reports an account of the overall RTD activity carried out in WP4, with a reference to tasks as set in the project (Annex 1). User requirements given by the SME during the first months (deliverable D2.1) are more explicitly addressed in sections 2 and 3, at the end of the technical overview.

Task T4.2: Methods for train profile reconstruction and load verification

Stated purpose:

Study and design of image analysis methods for extracting the profile of the train, in terms of loaded or empty wagons. Study, design and evaluation of global and local vision-based methods for the verification of the train load.

Results at month 18:

- o The pipeline for train profile reconstruction has been determined. Single modules of the pipeline have been tested, with positive results well below the objectives set by user requirements. The full pipeline has been tested on trains crossing the Vado Ligure plant.
- o At month 12, geometrical methods for the verification of the correct placement of containers on wagons, to be used in redundancy of those performed as part of the tasks of WP3, have been studied, developed and evaluated. Experimental analysis shows a sensitivity to noise superior to the objectives set by the SMEs in user requirements.

See Section 2 «Reconstruction of the train profile and automatic load verification».

Task T4.3: Ownership code identification

Stated purpose:

Implementation of automatic localisation and character recognition methods.

Results at month 18:

The pipeline for code character recognition has been designed, implemented and tested on the field on trains crossing the Vado Ligure plant.

See Section 3 « Localization and recognition of the ownership code ».

2. Reconstruction of the train profile

The final version of the train profile reconstruction software prototype is composed by a the following modules:

- Background removal on the train video sequence
- Computation of the train mosaic
- Rectangles and gaps detection on the mosaic

The remainder of the section reports implementation details.

Background removal

This module is based on the classical background estimation and change detection pipeline, typical of video analysis systems. The peculiarities of the specific application domain are:

- The dynamic event (i.e., train arrival or train departure) represents a considerable scene change, since its size is big compared to the whole viewed scene. This causes abrupt illumination changes due to video-camera compensation, in particular if the camera quality is low.

- The depicted scenario is outdoor, therefore there is a need for modeling background noise due to wind and temporary illumination changes (caused, for instance, by clouds passing by).

The implemented solution, based on codebooks [KCDD05], has been shown very effective for the application domain:

- *Codebook training*: the codebook model is constructed [KCDD05] on video sequences portraying the scene in the few minutes preceding the arrival of the convoy
- *Background removal*: according to the model built in the previous step, pixels in each frame are individually classified as background or foreground, and background pixels are deleted (set to black). In order to minimize the impact of background segmentation errors (due mainly to sudden changes in the scene illumination or of camera response - in case of auto-iris or auto-white-balance) on the subsequent steps, image regions known a priori to be part of the background (e.g, the sky region) are removed independently of adherence to the background model.

Mosaicing

A mosaic is built in order to obtain a unique complete image of the whole train. The adopted procedure exploits prior information on the nature of the train (a quasi planar object translating in front of the video-camera) and may be summarized as follows:

- Feature tracking
  - Select features to track via the Harris corner detector [HarrisStephens88]
  - Filter out from this set features that are adjacent to the background
  - Track features via the discrete version of the Lucas Kanade algorithm [LucasKanade81]
  - Filter out from the tracked feature those with no horizontal motion or a too big vertical displacement
- Image stitching
  - Estimate horizontal motion of the train by taking the median value of the horizontal displacement

In the case a small number of features is detected an ad hoc procedure, based on exploiting previous estimations of train velocity is applied.

Rectangle detection and selection

The final step consists of localizing rectangle shape objects in the mosaic image (that is, containers) and estimating the length of gaps (gaps between wagons, and empty spaces on wagons).

First we detect gaps between container (independently from their size), via the following algorithm:

- For each column (i.e., for each value of x in the image coordinates) in the train profile, count the foreground pixels.
- Segment continuous horizontal region for which the vertical pixel count does not exceed the maximum possible height for a loading deck, plus a little tolerance to account for noise in the background removal phase.
- The detected regions are considered load gaps.

The implemented method for rectangle detection may be summarized as follows:

- Extract lines from the mosaic image using Canny Edge Detection and Hough Line Detection in cascade. We recall here that the Hough Line detection algorithm returns line as a slope/intercept pair.

- Filter out all lines which are neither approximately vertical nor horizontal. "Approximately" here means we have given the algorithm a tolerance of a couple of sessagesimal degrees, in order to compensate for the fact that the image were taken by a hand-held camera (for stills), or on a camera on an unregistered tripod for movies.
- Mark in a separate list all vertical/horizontal lines that come from the edge between the foreground and the subtracted background.
- Keep in the line list only those lines which are a number of pixel approximately equal to a multiple of the container height (for horizontal lines) or width (for vertical lines) apart from a parallel line on the foreground/background border.
- Distances between parallel vertical lines are potential container lengths. Distances between parallel horizontal lines are potential container heights. Calculate them and build lists of the line pairs with the respective distance.
- For each admitted specification of container size:
  - Consider quartets made up of two parallel horizontal and two parallel vertical lines: they are a superset of all possible rectangles in the image. Keep only those whose aspect ratio and size are compatible (within a small tolerance) with the standard container dimensions.
  - Filter out the rectangles which are intersecting the background
  - For any cluster of heavily intersecting rectangles, choose the representative having the closest aspect ratio to the ideal one.
- If a bigger selected rectangle contains a pair of smaller ones, prefer the bigger rectangle over the pair if and only if the pair admits intersection.

Instead, load gap detection is implemented as follows:

- For each column (i.e., for each value of x in the image coordinates) in the train profile, count the foreground pixels.
- Segment continuous horizontal region for which the vertical pixel count does not exceed the maximum possible height for a loadinig deck, plus a little tolerance to account for noise in the background removal phase.
- The detected regions are considered load gaps.

Experimental results

Preliminary tests were carried out with a set of data acquired batch in two different circumnstances.

<u>**Batch tests**</u>

<u>BATCH test set 1 – Arquata Scrivia Station (IT)</u>

1. **Acquisition type:** Hand-held camera pictures of containers from convoys standing in Arquata Scrivia station.
2. **Data type:** Simple. The pictures are evenly illuminated due to cloudy weather.
3. **Content:** various containers at different distances
4. **Dataset size**: a few tenth of images

<u>BATCH test set 2 - Vado Ligure (IT)</u>

1. **Acquisition type:** High resolution video cameras mounted on a tripod.
2. **Data type:** Medium
3. **Content:** empty trains
4. **Dataset size:** 2 videos of approximately 3 minutes each.

The obtained results are in line with expectations.

- For what concerns batch test 1, being constituted by still images, we used it to test the system robustness to container detection, and all containers have been correctly localized.
- As for the videos :
  - o Background subtraction and mosaic construction: a qualitative analysis reports satisfactory results;
  - o rectangle detection: 1.5% errors
  - o gap estimation no errors

Field tests

The second and more consistent set of experiments was carried out on video-streams acquired on the field by the video-cameras currently installed at Vado Ligure (IT).

More details on the available video-cameras can be found in Deliverable D5.2. Here we just report a summary on the cameras appropriate for the our task:

1. CAM4 (the camera 4 reported in D5.2) – low-resolution video-surveillance camera previously installed in the plant for remote surveillance of the plant;
2. MEGA – a megapixel high resolution camera, installed mainly to perform tasks of WP5.

The data acquisition process started at month 16.

A maximum of two freight trains per week passes on the rail line, therefore the number of events of interest that may be recorded on site is limited. Luckily, the variability of weather conditions during the field tests allowed us to obtain a very rich and meaningful set of data. Table 1 reports a summary of the recorded videos, with comments on their features.

VideoID	**Camera**	**Weather**	**Train**	**artifacts**
M16-1-A	CAM4	cloudy	Empty	
M16-2-A	CAM4	sunny		
M16-3-A	CAM4	sunny	Empty	
M17-4-A	CAM4	sun – wind		
M17-5-A	CAM4	sun – wind	Train stops	
M17-6-A	CAM4	Cloudy		Overexp.
M17-7-A	CAM4	cloudy		Overexp.
M18-1-A	CAM4	Rain,mist		
M18-2-A	CAM4	rain		
M17-1-B	MEGA-far	Cloudy	Fast	
M17-2-B	MEGA-far	Very sunny		
M17-3-B	MEGA-far	Very sunny		Underexp.
M17-4-B	MEGA-close	sun – wind		
M17-5-B	MEGA-close	sun – wind	Train stops	
M17-6-B	MEGA-close	Cloudy		
M17-7-B	MEGA-close	cloudy	fast	
M18-1-B	MEGA-close	Rain,mist		
M18-2-B	MEGA-close	rain	fast	
M18-3-B	MEGA-close	Wind		

Table 1 – summary of the trains acquired during the field tests

A few comments on the data are in order

- The quality of pre-existing video surveillance cameras is very low, the camera is over-exposed and auto-iris makes it difficult to compute reliable backgrounds in difficult illumination conditions

- The Megapixel camera installation is prototypical and a more stable set up is needed. Thanks to the use of a codebook model, background subtraction is rather robust to the presence of wind (multiple codewords are associated to the same pixel to counter the changes due to camera oscillation). Instead wind breaks the assumption that relative motion between camera and train is translational, thus we may find artifacts in the constructed mosaic. Current software version implements a set of heuristics to counter this problem, but a more radical and effective solution would be a more stable camera installation (that is following the course of Metrocargo plant maintenance).
- Trains are crossing the plant, since it is installed by the side of a working rail from Savona Port to in-land. The ordinary week schedule includes two trains per week. Trains are not meant to stop in the area, even if they slow down since there may be men at work. We estimated average velocities around 15 Km/h, well within the user requirements. It was not possible to test the system at different speeds, since we can not influence on trains velocity.
- In spite of the fact that trains are rarely observed, the collected dataset is various enough to obtain significant evaluations: each train contains about [25-40] containers, all types of expected container types are present, some of them old and dirty, or otherwise blurred. Also, weather conditions have been very variable in the acquisition time span (see Table 1).

Results:

- Table 2 reports the results obtained for mosaic construction on videos acquired by the different camera types.
  - CAM4 was excluded from further experiments since the quality was too low (Fig 1)
  - MEGA-far was included in the evaluation although it has been decided that the level of zoom was not appropriate for other operations (in particular code reading)
  - MEGA-close returns good results (see Fig 2) and the final parameters configuration found will be kept for future experiments. MEGA-close is divided in two groups since a parameter tuning occurred in the middle.

Camera Type	**Qualitative evaluation**
CAM4	Low-quality cameras are rather weak to background construction and feature tracking. The obtained mosaics are often ruined with artifacts (see Figure 1)
MEGA-far	Good background construction and change detection (even in case of wind). Robust feature tracking. Very strong wind affects mosaics quality, otherwise the obtained results are quite good.
MEGA-close (month 17)	As above. The presence of more high frequency details required a new system configuration. Occasional faults of the camera (high sensitivity to contrast adjustment and video-compression errors) caused errors to the mosaic construction, otherwise results are good
MEGA-close (month 18)	As above. A camera tuning improved the quality of the signal, thus results are usually satisfactory

Table 2 – Qualitative analysis of mosaic construction steps

- Table 3 reports the obtained results on containers and gaps detection, highlighting the fact that the final camera configuration is appropriate for the problem and the obtained results are well below the user requirements (example images in Figures 3 and 4).

Camera Type	**Containers error % (average)**	**Gaps error % (average)**
MEGA-far	22%	8.4%
MEGA-close (month 17)	20.5%	3%
MEGA-close (month 18)	3.9%	0%

Table 3 – Quantitative analysis of containers and gaps localization

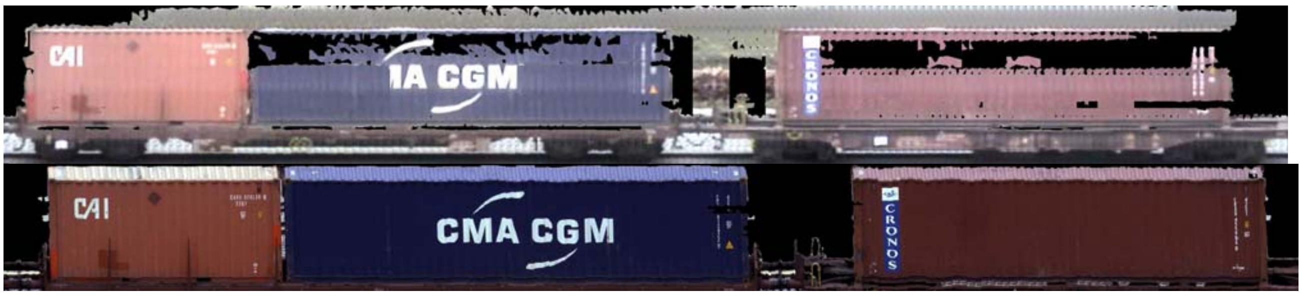


Figure 1 – Comparison between CAM4 (top) and MEGA (bottom) on a rather difficult situation (bright sun and wind) speaks in favor of the Megapixel technology

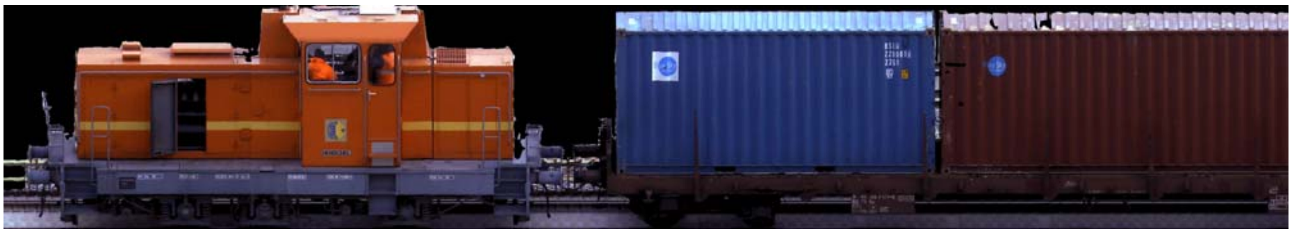


Figure 2 – A detail of a constructed mosaic from a Megapixel video (notice the background removed and the quality of the obtained stitch)

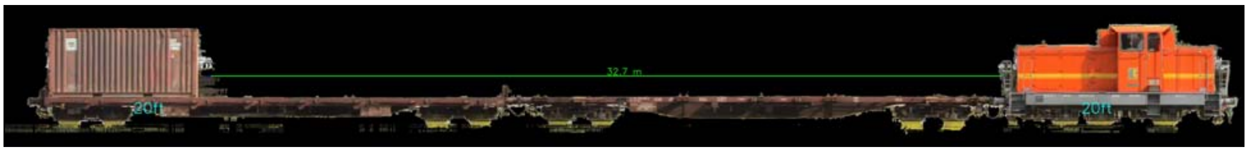


Figure 3 – Containers and gaps detection on a small train

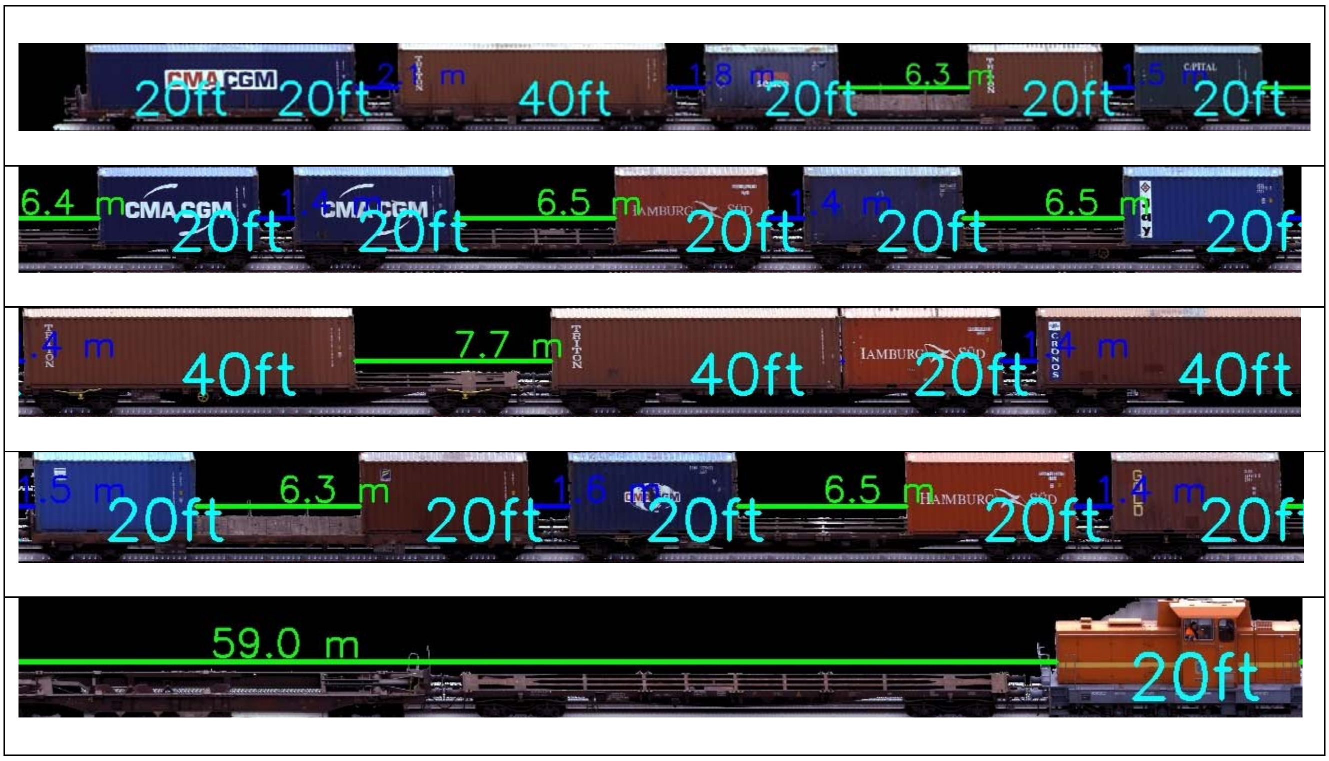


Figure 4 – Chuncks of a long video containing details on containers and gaps detection

Conclusions

By user requirement:

- o **Reconstruct the train profile (as a sequence of empty or filled wagons)**
  - The field tests underlined that megapixel camera is appropriate for the task. Satisfactory results were obtained under various weather conditions. The reconstructed trains include all kinds of containers, and various configurations of filled and empty wagons.
- o **Verifying that each container or swap body is correctly loaded, i.e. that all retaining blocks fit correctly the relevant corner fitting and that the container has been fully lowered on the block.**
  - Laboratory tests carried out before month 12 lead us to conclude that the reliability level of such operations based on video signals acquired from a distance is not high enough to meet user requirements. Conversely, field tests carried out within WP3 showed how, a close range view on the corner fittings is effective for such tasks, to the extent that a redundant solution is not needed.
- o **Generate a recording of the sequence of containers on the trains, including the information on their size (20’, 30’, 40’, 45’)**
  - The recording is generated and saved both on textual log files and on multi-media material (the annotated mosaic image).
- o **Reconstruction can be done either whilst the train is coming to a stop in the station (in this case in initial speed of 60 km/h with 1 m/sec2 deceleration should be considered) or when it is standing in the station**
  - The implemented solution assumes that reconstruction is performed as the train enters or leaves the station. The hypothesis of motion is used to counter the fact that with the currently installed cameras the whole train would not be visible while it is standing in the station. The developed software modules would be easily adapted (by changing the mosaic construction module) in case of different plant configurations.
  - For what concerns the trains speed, we had no influence on the speed of trains passing by the plant. Most of them decelerated in the plant area (since there where humans operating in the neighbourhood), and only one of them stopped. The estimated speeds are about 15 Km/h.

By task goal:

- o **T4.2 - Study and design of image analysis methods for extracting the profile of the train:**
  - Effective solutions for the problem under analysis have been studied, developed and tested. A software prototype performing these tasks, given a video stream as a input is a part of present deliverable.
- o **T4.2 - Study, design and evaluation of global and local vision-based methods for the verification of the train load.**
  - The present feasibility study underlined the fact that the proposed solutions are not effective for the problem.

By measurable objective:

The measurable objective set by the SME's for this task in the user requirements (deliverable D2.1) is as follows:

- o **Maximum error percentage - reconstruction of the train profile: 1 position per train of 22 wagons equals 4.5% over the total number of observed wagons in standard conditions, 2 positions equals 9% in extreme conditions. An error is observed either if an empty slot of a wagon is missed, or a container size is wrongly associated.**
  - The final parameter setting for Megapixel camera allows us to achieve good results, below the indicated measurable objectives, on a set of video-streams acquired in various (and sometimes rather challenging) weather conditions

3. Localization and recognition of the ownership code

The module prototype for localization and recognition of ownership codes has not been subjected to significant changes with respect to what had already been described in deliverable D4.1, since its performance has already been found to be fully compliant to the workpackage objectives at month 12. Therefore, all that is written in deliverabe D4.1 still holds, and the reader could as well refer to that.

A final summarization step, considering multiple occurrences of the same code over the image sequence has been added.

Anyway, we will summarize the prototype architecture and its performance on an experimental dataset again in this deliverable, for ease of reference.

- o Segment the image into connected components using Niblack's algorithm [Niblack85]
- o Filter out connected components if:
  - the bounding box is outside the acceptable size range
  - the aspect ratio [ZhuQiJiangXu07] is outside an acceptable range
  - filter out connected components with low edge contrast [ZhuQiJiangXu07]
  - for each connected component compute an appropriate feature vector that includes shape and contrast information (see Deliverable D4.1 for details.
- o Feed the resulting feature vector to an appropriately trained RLS classifier to classify text elements.
- o Combine text elements in text blocks by computing connected components on the characters graph:
  - characters are the graph nodes; graph edges connect two characters of similar height and whose distance is lower than a threshold related to their height
- o For each text block
  - Compute the text orientation
  - Compose the code string according to the computed text orientation
  - Read each character of the string with an OCR system built via a OVA SVM classifier.
  - Align the resulting string to the expected one via the Needleman-Wunsch [NW70] algorithm.
  - The distance between two strings (the expected and the estimated) is computed following a variant of the Lehvenstein distance:
    - ▪ Gaps in the read string contribute a value of one-half
    - ▪ Gaps in the expected string are ignored
    - ▪ Proper character mismatches contribute a value of one.
- o The image is associated to a code if the computed distance does not exceed 3

Each frame of the video sequence is processed according to the above pipeline. To each frame we associate at most one container code. A code is read if, considering all the frames associated to the expected code, all the expected characters have been correctly read at least twice.

Experimental results

Batch tests (image based)

BATCH test set 1 – Various locations

- **Acquisition type:** different locations (mainly containers repository areas) in various weather and illumination conditions (heavy rain, shadow, direct sun) and with various input devices (different photo- and video-cameras).
- **Data type:** medium/difficult. Images acquired by the RTD performers, mostly high resolution
- **Content:** various containers at different distances
- **Dataset size**: 237 images.

The dataset has been collected to train data-driven modules (character detection, and OCR). Thus the performed tests can provide us only with a separate *cross-validation* evaluation of the error for each module:

- 2% for detection,
- 3.4% for character recognition.

We remark that given a more extensive dataset, in particular for character recognition, both modules are capable of performing better.

We can however combine those error estimates into an estimate on the whole pipeline by a stochastical simulation of the whole pipeline, in which errors are randomly introduced with the probabilities estimated above. Performing such a simulation on a million container codes led the system to:

- mistakenly reject a correct container code in 0.02% of cases (false negatives),
- to mistakenly accept a wrong container code in no observed case (false positives).

BATCH test set 2 – Images of Vado Ligure supplied by SMEs

- **Acquisition type:** close range image sequences of trains
- **Data type:** medium. During acquisition the scene was very well illuminated, shots are taken close range (indeed, these images could not be used for Task 4.2 as the containers edges are not visible) thus container codes are rather big. Images supplied by the SME's.
- **Content:** 2 trains at a fixed distance.
- **Dataset size**: 162 images

Being these images totally unrelated to the dataset a proper evaluation was possible:

- False negatives: the system mistakenly rejected one code (0.6% of the total).
- False positives: none.

Field tests (video-based)

Field tests were performed on the video-streams acquired at Vado Ligure (See table 1). A few comments on the available data are in order:

- The quality of low-resolution video-surveillance camera is too low (codes are invisible even to an human observer) therefore our analysis is limited to MEGA. For similar reasons, the set referred to as MEGA-far in Table 1 is again unuseful for the task.
- The set referred to as MEGA-close is the most appropriate for the task, although a further zoom in (which would be possible with the installed camera) will help enlarging the characters to be read and simplifying the segmentation phase (indeed, current settings have been chosen as a compromise between WP4 and WP5 requirements. An additional camera will solve this issue.
- Since the amount of data available is limited we relied on the same training and validation set used in previous analysis. The average size of characters on the training set is much bigger (of at least one third) than the ones visible on test data.

The analysis of errors is based on the following considerations:

- Container codes have been manually annotated to simulate the anticipated load plans. This information allows us to estimate the number of false negatives (ie., container codes belonging to the anticipated load plan and missing in the code verification)
- To deal with false positives, being in a verification problem codes belonging to the anticipated load plan by mistake and erroneously verified in the verification phase we simulated 10.000 random wrong codes for each container and tried to verify them (the procedure is similar to the one followed during laboratory tests)

On the basis of these experimental specifications the obtained results are (average):

- False positives 0.042%
- False negatives 10.2%

At the beginning of month 18 the Megapixel camera has been tuned to improve its sharpness. The results obtained in this limited time span are (on average):

- False positives 0.01%
- False negatives 4.3%

The weather conditions encountered during most of test sessions classify most of the situations has hard or extreme.

The obtained results are rather satisfactory but there is also some space for improvement:

- During the live test sessions, due to the limited amount of trains, it was not possible to gather an ad hoc dataset for the data-driven steps (text detection and OCR). A more specific dataset would guarantee an improvement in the obtained results. Data are currently being gathered at this purpose. We estimate that, considering the fact trains are rare but rather long, in two months from the end of the project and ad hoc dataset will be available
- A further zoom in of the camera and the consequent enlargement of text elements would improve the quality of the obtained segmentation (see Fig. 5) and, as a consequence, would lead to a better recognition. This is confirmed by batch data acquired under those conditions that lead to a very low error rate.

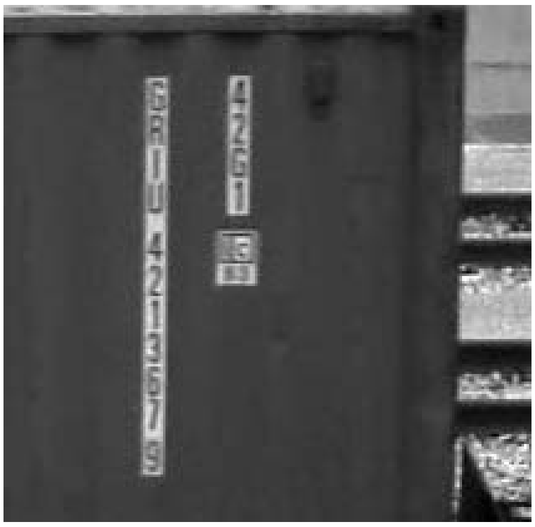
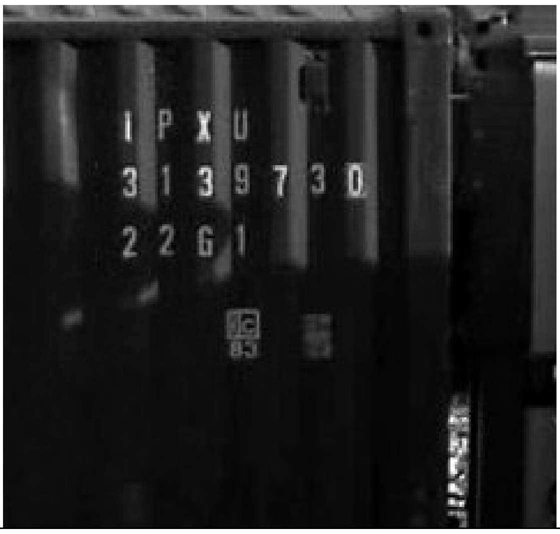
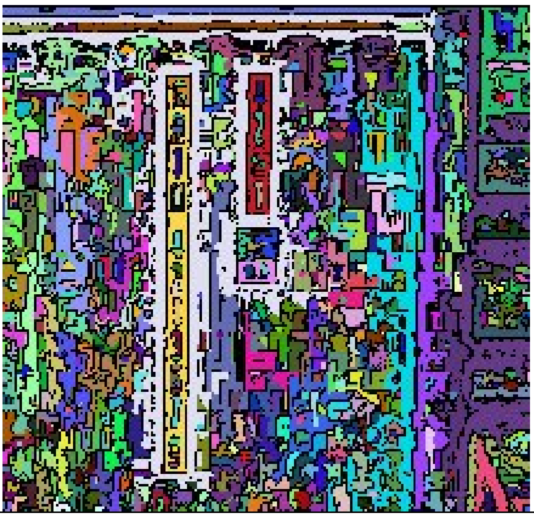
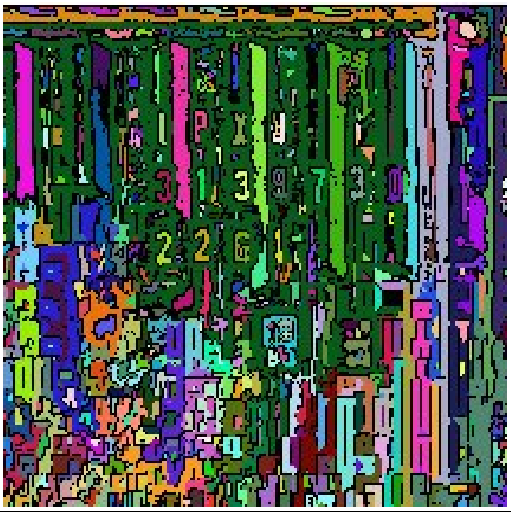
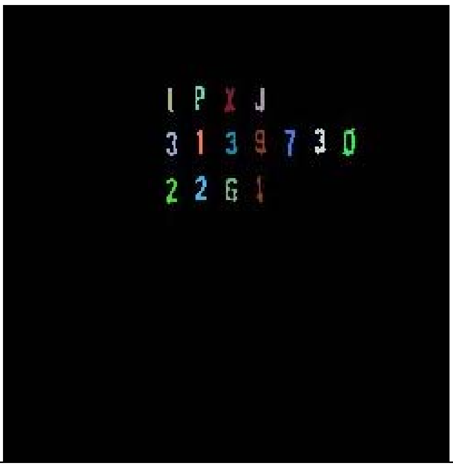
	
Original image (detail)	
	
Connected Components computed via segmentation	
	
Localized characters	

Figure 5 – Preprocessing phase of the OCR module. On the left column a vary bad quality image, produces a bad segmentation and a failure in the text localization module. With good quality image (right) all task are correctly accomplished.

Conclusions

By user requirement:

- o **Read and check against the foreseen load plan the ownership codes (alphanumeric strings) painted on the side of each container. The painting may be faded and damaged, per sample pictures supplied by SMEs**
- o **Checking the ID and position of load units versus the anticipated load plan**
- o **Verifying the correct loading of the containers in the foreseen positions, checking their ID**
- o **Verifying the correct loading of the containers in the foreseen positions of outgoing trains, checking their ID by 2D train scanning. Due to the high level of accuracy of control on the container location within the Metrocargo plant (practically zero error), this verification is actually redundant and it is acceptable that it be done while train is leaving the station. Any mistake thus detected will be dealt with at the following stop of the train.**
  - The field tests underlined that the megapixel camera is effective for the task. Satisfactory results were obtained under various weather conditions. The reconstructed trains include all kinds of containers, different levels of damage and dirt, container codes located in different positions, printed with different fonts and colors.

By task goal:

- o **T4.3 – Ownership code identification**
  - Effective solutions for the problem under analysis have been studied, developed and tested. A software prototype performing these tasks, given a video stream as a input is a part of present deliverable.

By measurable objective:

- o **Maximum error percentage on incoming train – automatic code reading: 1 container out of 50 equal 2% in standard conditions, 2 containers equal 4% in extreme conditions. A code is erroneously read if it will not match with the anticipated load plan, leading to a remote human intervention, and the load plan is proven correct. A direct human interventions is admitted if the visible ownership code is damaged to be unreadable (it is not measurable).**
- o **Maximum error percentage on outgoing train --- automatic code reading: 1 container out of 50 equal 2% in standard conditions, 2 containers equals 4% in extreme conditions. A code is erroneously read if it will not match with the load plan of the Metrocargo plant, leading to notification to the general control system, to human verification and subsequent action at the following station.**
  - The results obtained on the last batch of field tests in sub-optimal conditions (mostly depicting hard or extreme weather conditions) are within the required percentages. Also, a further zoom in of the Mega-pixel camera will allow for further improvements (as demonstrated by previous batch experiments carried out on zoomed images distributed by SME's). The available camera allows for this zoom, but current settings of the optics have been chosen as a compromise with WP5 requirements. An additional camera will solve this limitation.

4. Description of the software prototype

Here is a description of the software prototype result of the feasibility study. We organize it in:

- o Mosaicing suite
- o Containers and gaps estimation suite
- o OCV suite

Mosaicing suite

bgremove_tool

```
Usage: ./bgremove_tool [options] <input files>

Options:
  --help                    produce help message
  --skip arg (=0)           Skip first n frames of the input files
  --start arg (=0)          Start processing at given second of the video
  --end arg (=-1)           End processing at given second of the video
  --mframes arg (=300)      Number of frames used in building the bg model
  --seg-only                Use opencv fg segmentation for foreground
                            detection
  --both-masks              Use both opencv fg segmentation and raw output
                            masks. Creates two output for each input file
  --smalloutput             Output frames at the same size of the ROI derived
                            from roofline and groundline
  --roofline arg (=0)       All pixels above this y value are considered bg
  --groundline arg (=-1)    All pixels below this y value are considered bg
  --leftline arg (=0)       All pixels left of this x value are considered bg
  --rightline arg (=-1)     All pixels right of this x value are considered bg
  --aggressiveness arg (=1) Aggressiveness modifier for the bg subtractor
  --verbose                 be more verbose
```

Program description

This program is a proof-of-concept application written in order to check and demonstrate the feasibility of codebook-based background subtraction in the train profile reconstruction pipeline. A good quality background subtractor helps in feeding a better signal-to-noise ration in the subsequent (mosaicing and gap/container detection) modules. In the current implementation of the pipeline, gap detection is based exclusively on the classification as foreground or background of pixels in the profile, so functioning of the background subtraction is necessary.

The program takes as input any number of .avi files, which get all processed according to the same parameters, set via the command-line options. It outputs a black background, optionally cropped (see description of the --smalloutput option) version of the input files, with .nobg.avi (by default) or .nobg-segm.avi (see optiions --seg-only and --both-masks) appended to the filename.

The algorithm on which the module is based is already described in sections above. Here only the interface of the proof-of-concept executable will be documented.

Options description

--help: print a help message and exit.

--skip *arg*: Skip first *arg* frames of the input files. This in addition to any skipping specified via the **--start** option; that is bgremove_tool --skip 3 --start 9 will start building a background model after nine seconds and three frames of the input video. *arg* defaults to 0.

--start *arg*: Skip first *arg* seconds of the input files. This in addition to any skipping specified via the **--skip** option; that is bgremove_tool --start 3 --skip 9 will start building a background model after three seconds and nine frames of the input video. *arg* defaults to 0.

--end *arg*: End processing at the second of the video specified by *arg*. This is used to clip the output after the train has passed (in an integrated enviroment, this will be probably replaced by a signal sent to the module by the system). The default value for *arg*, -1, causes the system to process the input videos till their end.

--mframes *arg*: Number of frames used in building the bg model. Once a background model is built, the system starts actual removal from subsequent frames. *arg* defaults to 300.

--seg-only: Use opencv fg segmentation on the foreground mask. This gives a foreground mask with less "holes". When this option is given, the output file suffix turns from .nobg.avi into .nobg-segm.avi.

--both-masks: Use both opencv foreground segmentation and raw output masks. Creates two output files for each input file, with .nobg-segm.avi and .nobg.avi suffixes respectively.

--smalloutput: Output only the content of the rectangle of interest (ROI) specified by the **--roofline**, **--groundline**, **--leftline**, and **--rightline** options. If this option is given, frames in the output files will have the same size as the ROI. If this option is not given, frames in the output file will have the same size as frames in the input files, and the area outside the ROI will be classified as background regardless of the model built.

--roofline *arg*: Specifies the upper y coordinate of the ROI. All pixels above this y value are automatically considered background. *arg* defaults to 0, meaning the upper border of the ROI will coincide with the top of the image.

--groundline *arg*: Specifies the lower y coordinate of the ROI. All pixels below this y value are automatically considered background. *arg* defaults to -1, meaning the lower border of the ROI will coincide with the bottom of the image.

--leftline *arg*: Specifies the leftmost x coordinate of the ROI. All pixels left of this x value are automatically considered background. *arg* defaults to 0, meaning the leftmost border of the ROI will coincide with the leftmost border of the image.

--rightline *arg*: Specifies the rightmost x coordinate of the ROI. All pixels right of this x value are automatically considered background. *arg* defaults to -1, meaning the rightmost border of the ROI will coincide with the rightmost border of the image.

--aggressiveness *arg*: Aggressiveness modifier for background subtraction. *arg* must be a positive number. Higher values will cause the subtraction to be performed more aggressively, i.e. more pixels will be classified as background. This is implemented by actually multiplying by *arg* the size of the interval around a background model codeword in which a pixel is considered to fit the model . Consistently with this semantics, *arg* defaults to 1.

--verbose Give a more verbose output to the console.

Parameters used in experiments

All experiments described in this report were performed with the **--seg-only** and **--smalloutput** flags set.

Three different setups, corresponding to the three different camera setups in the dataset, were used for the **--roofline**, **--groundline**, **--leftline**, **--rightline** and **--aggressiveness** parameters. Those values have been keep fixed inside each dataset, simulating a fixed system setup.

In particular:

- **CAM4**: `--roofline 50 --groundline 320 --leftline 50 --rightline 530 --aggressiveness` **???**
- **MEGA-far**: `--roofline 100 --groundline 442 --leftline 520 --rightline 936 --aggressiveness` **???**

- **MEGA-close**: `--roofline 80 --groundline 570 --leftline 500 --rightline 960 --aggressiveness` **???**

Each single video has instead its own values for --mframes and --end, depending on the amount of available background footage and the time taken by the train passing.

Usage example

As an example, the included video M18-2-B, can be succesfully processed via the following command line:

```
./bgremove_tool --aggressiveness 0.9 --roofline 80 --groundline 570 --
leftline 500 --rightline 960 --smalloutput  --mframes 700 --end 259 --seg-
only M18-2-B.avi
```

Yielding a background-removed version called M18-2-B.avi.nobg-segm.avi.

mosaictrain

```
Usage: ./mosaictrain [options] <input files>

Options:
  --help                produce help message
  --novideo             suppress video output
  --nostop              don't stop for debugging purposes
  --startpos arg (=0)   Starting position in seconds
  --endpos arg (=-1)    Stopping position in seconds
  --roofline arg (=0)   Don't track features above this y value
  --groundline arg (=-1) Don't track features below this y value
  --leftline arg (=0)   Don't track features left of this x value
  --rightline arg (=-1) Don't track features right of this x value
  --pptfu arg (=-1)     Lenght of a TFU in pixels. Activates train velocity
                        estimation.
  --verbose             be more verbose
```

Program description

This program is a proof-of-concept application written in order to check and demonstrate the feasibility of codebook-based background subtraction in the train profile reconstruction pipeline. Bad quality mosaicing could lead to the failure of the whole pipeline, yielding reconstructed images of the train profile which are not recognizable even by human vision.

The program takes as input any number of .avi files, which get all processed according to the same parameters, set via the command-line options. It outputs one image of the whole of the train for each video, named as the original file with the suffix .patchwork.png appended. By default it also optionally displays, in real time, the tracking and mosaicing process, for debugging purposes. In this case, it stops automatically on frames he recognizes as problematical to track, unless given the --nostop option. If given a measure of conversion from pixel to metres (via the --pptfu option) it displays in the tracking window also an estimate (in km/h) of the train velocity.

This program yields better results on videos on which the background has already been removed, for instance via bgremove_tool.

The algorithm on which the module is based is already described in previous sections. Here only the interface of the proof-of-concept executable will be documented.

Options description

--help: print a help message and exit.

--novideo: Do not display the tracking and mosaicing process as it is performed. This flag is meant for batch operation, and also implicitly includes **--nostop**

--nostop: Don't wait for the user to press a key each time the program encounters a particularly problematic frame. This option is implicit in **--novideo**.

--startpos *arg*: Start processing the video at the second specified by *arg*. Default is zero, meaning the video is processed from the start.

--endpos *arg*: Stop processing the video at the second specified by *arg*. Default is minus one, meaning the video is processed until the end.

--roofline *arg*: Do not look for features to track above the y value specified by *arg*. Default is zero, meaning the upper limit to feature search is the top boundary of the frames.

--groundline *arg*: Do not look for features to track below the y value specified by *arg*. Default is minus one, meaning the lower limit to feature search is the bottom boundary of the frames.

--leftline *arg*: Do not look for features to track left of the x value specified by *arg*. Default is zero, meaning the leftmost limit to feature search is the left boundary of the frames.

--rightline *arg*: Do not look for features to track above the y value specified by *arg*. Default is zero, meaning the upper limit to feature search is the top boundary of the frames.

--pptfu *arg*: Specify the lenght of a TFU (20' unit) in pixels. The 20' lenght has been chosen because it can be easily measured directly from a video snapshot. Giving this option activates train velocity estimation. The default (meaning "unknown, do not convert from pixels to real lenghts") is minus one.

--verbose Yields verbose output to the console.

Parameters used in experiments

All experiments described in this report were performed with the **--seg-only** and **--smalloutput** flags set.

All experiments described in this report were performed with the **--novideo** flag set. This only for a matter of convenience (batch processing), as the **--novideo** flag has no influence whatsoever on the obtained image.

Three different setups, corresponding to the three different camera setups in the dataset, were used for the **--roofline** and **--groundline** parameters, in order to limit the tracking of perspective-deformed features on the container roofs and static features on the ground, respectively. Those values have been keep fixed inside each dataset, simulating a fixed system setup.

In particular:

- **CAM4:** `--roofline 30`
- **MEGA-far:** `--pptfu 537 --roofline 60`
- **MEGA-close:** `--pptfu 675 --roofline 60 --groundline 400`

No parameters have been changed at the single video level.

Usage example

As an example, the video M18-2-B.avi.nobg-segm.avi, obtained from included video M18-2-B.avi via the command line in the last example, can be turned in a high-resolution image of the whole convoy* by giving the following command:

```
./mosaictrain --pptfu 675 --roofline 60 --groundline 400 --nostop M18-2-
B.avi.nobg-segm.avi
```

Yielding an image of the train in the file M18-2-B.avi.nobg-segm.avi.patchwork.png.5.

Containers and gaps estimation

Findcont.pyc

```
Usage: findcont.pyc [options] imagelist

Options:
  -h, --help            show this help message and exit
  -t CANNY_LOW, --canny-low=CANNY_LOW
                        Canny low threshold (default 50)
  -T CANNY_HIGH, --canny-high=CANNY_HIGH
                        Set canny high threshold to CANNY_HIGH (default 200)
  -H HOUGH_THRESH, --hough-thresh=HOUGH_THRESH
                        Set hough threshold to HOUGH_THRESH (default 100)
  -W CWIDTH, --cont-width=CWIDTH
                        Approximate (+- 5%) container width in pixel. Default
                        is zero, meaning that the program will ask just for a
                        correct aspect ratio.
  --cont-height=CHEIGHT
                        Approximate container height in pixel. To be manually
                        set only for prospectical calibration, otherwise it
                        will be automatically determined.
  -r ROOFLINE, --roofline=ROOFLINE
                        y coordinate of the top line of the convoy ("container
                        roofs")
  -g GAP_HEIGHT, --gap-height=GAP_HEIGHT
                        Equivalent in metres of vertical foreground pixels for
                        gap detection
  --two-hough           Use separate hough filters for vertical and horizontal
                        line detection.
  --with-hc             Look also for high cube containers in the convoy.
```

Program description

This program is a proof-of-concept application written in order to check and demonstrate the feasibility of rectangle-detection-based container localization and background-removal based gap detection for the train profile reconstruction pipeline, of which it constitutes the last module.

The program takes as input any number of image files (in any format supported by opencv), which get all processed according to the same parameters, set via the command-line options. It outputs an annotated, human readable version image: near each detected container it writes the type (out of 20 feet normal, 40 feet normal, 40 feet high-cube and 45 feet high-cube). Gaps between container are marked by continuous lines, and an estimation of their lenght is shown. Gaps big enough to fit at least one 20 feet container are highlighted in green, while the other ones are marked in blue.

The output image filename is obtained by appending the suffix .profile.jpg to the filename (and path) of the original image.

The algorithm on which the module is based is already described in section XXX. Here only the interface of the proof-of-concept executable will be documented.

Options description

--help: print a help message and exit.

-h, **--help**: show a help message and exit

-t *arg*, **--canny-low**=*arg*: Set the low threshold of the Canny edge detector to *arg*. Default value is 50

-T *arg*, **--canny-high**=*arg*: Set the low threshold of the Canny edge detector to *arg*. Default value is 200.

-H *arg*, **--hough-thresh**=*arg*: Set the threshold for the Hough transform line detector to *arg*. Default value is 100

-W *arg*, **--cont-width**=*arg*: Assume that *arg* is the approximate (+- 5%) width in pixel for a twenty feet container. Default value is zero, meaning that the program will look just for a correct aspect ratio.

--cont-height=*arg*: Assume that *arg* is the approximate height in pixel for a twenty feet container. To be manually set only for prospectical calibration, otherwise it will be automatically determined.

-r *arg*, **--roofline**=*arg*: Assume that *arg* is the y coordinate of the top line of the convoy ("container roofs"), and use this assumption for container selection.

-g *arg*, **--gap-height**=*arg*: Equivalent in metres of vertical foreground pixels for gap detection. Pixels columns in which at least *arg* metres worth of pixels have not been classified as background will not be classified as part of a gap between containers.

--two-hough Use separate hough filters for vertical and horizontal line detection. The threshold of the vertical line detector will be proportional to the horizontal one via the container aspect ratio.

--with-hc Look also for high cube containers in the convoy. By default, the system looks only for normal height containers.

Parameters used in experiments

All experiments described in this report were performed with the parameter -g set to 2.9.

Three different setups, corresponding to the three different camera setups in the dataset, were of course used for the parameters specifying the 20 feet container height and width in pixels, due to the different distances/zoom levels by which the cameras look at the train. Those values have been keep fixed inside each dataset, simulating a fixed system setup.

In particular:

- **MEGA-far**: `--cont-width=537`
- **MEGA-close**: `--cont--width=675 --cont-height=295`

No parameters have been changed at the single video level.

Usage example

As an example, the train mosaic M18-2-B.avi.nobg-segm.avi.patchwork.png, obtained from included video M18-2-B.avi via the command lines described in the previous examples, can successfully processed by the following command:

```
./python findcont.pyc --pptfu 675 --roofline 60 --groundline 400 M18-2-B.avi.nobg-segm.avi.patchwork.png
```

Yielding an annotated image of the train in file M18-2-B.avi.nobg-segm.avi.patchwork.png.profile.jpg

OCV suite

check-codes

```
Usage: ./check-codes [options] ImageList outputfile

Where:
   ImageList:  file of rows of the format "<imagename> <expectedstring>"
   outputfile: output filename. The output file will be written in the same
               format of ImageList

Options:
  --locScale arg (=Modelli/scaleLoc)    set custom scale file for character
                                        localization
  --ocrScale arg (=Modelli/scaleOCR)    set custom scale file for character
                                        recognition
  --locModel arg (=Modelli/modelLoc)    set custom model file for character
                                        localization
  --ocrModel arg (=Modelli/modelOCR)    set custom model file for character
                                        recognition
  --minCharSize arg                     set minimum size of expected characets. -
                                        1 disable all filters based on size
  --maxCharSize arg                     set maximum size of expected characters.
                                        -1 disable all filters based on size
  --textdth arg                         threshold for text detection.
  --charSize arg                        mean expected char size used to set
                                        internal parameters
  --readall                             read also images for which we are not
                                        expecting a container code
  --help                                produce help message
```

Note: for each option is provided a default value and for the first four options with the executable is delivered also a file that can be used as parameter.

Program description

The program take in input a sequence of video frames and container codes and check if in a given frame is present the expected code.

The program need in input a file with the list of images and their corresponding codes formatted with a sequence of rows of the form

<imagename> <expectedstring>

and generate an output file with the readed characters for each image. Each readed text block is separated by a white space.

Usage example

```
./check-codes input-file output-file --maxCharSize 20 --minCharSize 10 --
textdth -1.5 --charSize  15
```

5. Related bibliography

[Chan01] Y.K. Chan and C.C. Chang. Image matching using run-length feature. Patt. Recogn. Letters, 22(5), 2001

[Niblack85] W. Niblack. An introduction to digital image processing. Strandberg Pub. Company, Denmark, 1985.

[ZhuQiJiangXu07] K. Zhu, F. Qi, R. Jiang, and L. Xu. Automatic character detection and segmentation in natural scene images. Journal of Zhejiang University-Science A, 8(1):63–71, 2007.

[ZhangLu04] D. Zhang and G. Lu. Review of shape representation and description techniques. Patt. Rec., 37(1):1–19, 2004.

[ZDO09] L. Zini, A. Destrero and F. Odone. A classification architecture based on connected components for the detection of text in unconstrained environments. *Proceedings of the 6th IEEE International Conference on Advanced Video and Signal Based Surveillance,* 2009

[KCDD05] K. Kim, T. Chalidabhongse, H. David and L. Davis, Real-time foreground–background segmentation using codebook model, *Real-Time Imaging* **11** (2005*)*

[HarrisStephens88] C. Harris and M.J. Stephens. A combined corner and edge detector. Alvey Vision Conference, pages 147–152, 1988.

[LucasKanade81] Lucas B D and Kanade T 1981, An iterative image registration technique with an application to stereo vision. *Proceedings of Imaging understanding workshop*, pp 121—130

[NW70] Needleman SB, Wunsch CD. (1970). "A general method applicable to the search for similarities in the amino acid sequence of two proteins". J Mol Biol 48 (3): 443–53.