Grant Agreement Number: 687458

Project acronym: INLANE

Project full title: Low Cost GNSS and Computer Vision Fusion for Accurate Lane Level Navigation and Enhanced Automatic Map Generation

D2.3

Report on developed vision-based software modules v1

Due delivery date: 31/12/2016
Actual delivery date: 30/12/2016

Organisation name of lead participant for this deliverable: VICOM

<table>
<thead>
<tr>
<th>Dissemination level</th>
<th>PU</th>
<th>PP</th>
<th>RE</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Public</td>
<td>Restricted to other programme participants (including the GSA)</td>
<td>Restricted to a group specified by the consortium (including the GSA)</td>
<td>Confidential, only for members of the consortium (including the GSA)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This document presents the first report on developed vision-based software modules. It describes and clarifies the work done over the course of the first 12 months of the project. The modules described in this deliverable are aligned with the information already provided in the Description of Action for INLANE (as per Grant Agreement number 687458).
Legal Disclaimer

The information in this document is provided “as is”, and no guarantee or warranty is given that the information is fit for any particular purpose. The above referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law. © 2016 by INLANE Consortium.
### Abbreviations and Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>FDR</td>
<td>False Discovery Rate</td>
</tr>
<tr>
<td>FNR</td>
<td>False Negative Rate</td>
</tr>
<tr>
<td>fps / FPS</td>
<td>Frames per Second</td>
</tr>
<tr>
<td>GNSS</td>
<td>Global Navigation Satellite System</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>GSA</td>
<td>European GNSS Agency</td>
</tr>
<tr>
<td>H2020</td>
<td>Horizon 2020</td>
</tr>
<tr>
<td>HOG</td>
<td>Histogram of Oriented Gradients</td>
</tr>
<tr>
<td>RAM</td>
<td>Random-Access Memory</td>
</tr>
<tr>
<td>RANSAC</td>
<td>Random Sampling Consensus</td>
</tr>
<tr>
<td>ROI</td>
<td>Region of Interest</td>
</tr>
<tr>
<td>SLAM</td>
<td>Simultaneous Localization and Mapping</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>TRL</td>
<td>Technology Readiness Level</td>
</tr>
<tr>
<td>TSR</td>
<td>Traffic Sign Recognition</td>
</tr>
<tr>
<td>VO</td>
<td>Visual Odometry</td>
</tr>
<tr>
<td>WP</td>
<td>Work Package</td>
</tr>
</tbody>
</table>
# Table of Contents

1. Executive Summary ......................................................................................................................... 9  
2. Introduction .................................................................................................................................... 10  
   2.1 Purpose of Document ........................................................................................................... 10  
   2.2 Intended Audience ................................................................................................................ 10  
3. Lane detection ............................................................................................................................... 11  
   3.1 Design ................................................................................................................................... 11  
   3.2 Implementation ...................................................................................................................... 14  
   3.3 Results .................................................................................................................................. 15  
   3.3.1 Videos evaluated ............................................................................................................... 16  
   3.3.2 Performance evaluation .................................................................................................... 16  
   3.3.3 Computational cost ............................................................................................................ 23  
   3.3.4 Subjective analysis ............................................................................................................ 24  
4. Traffic Sign Recognition ................................................................................................................. 28  
   4.1 Design ................................................................................................................................... 28  
   4.1.1 Detection ........................................................................................................................... 29  
   4.1.2 Tracking............................................................................................................................. 33  
   4.1.3 Classification ..................................................................................................................... 35  
   4.1.4 Tracking-classification validation strategy ......................................................................... 38  
   4.1.5 Real-world location of classified traffic signs .................................................................... 40  
   4.2 Implementation ...................................................................................................................... 44  
   4.3 Results .................................................................................................................................. 45  
   4.3.1 Performance accuracy ...................................................................................................... 45  
   4.3.2 Computational load ........................................................................................................... 46  
5. Visual Odometry ............................................................................................................................ 48  
   5.1 Design ................................................................................................................................... 51  
   5.2 Implementation ...................................................................................................................... 51  
   5.3 Results .................................................................................................................................. 53  
6. Conclusions ................................................................................................................................... 56  
7. References ..................................................................................................................................... 57
List of Figures

Figure 1: Block diagram of the lane detection algorithm .................................................. 12
Figure 2: Lane markings segmentation ........................................................................... 13
Figure 3: Example histogram of x-positions ..................................................................... 13
Figure 4: Kalman filtered road model: in white, the detected points, in blue, the lane model predicted with the Kalman Filter: (left) straight road; (right) curvy road ........................................... 14
Figure 5: Lane markings manual annotation with Viciprecs tool ..................................... 17
Figure 6: Synthetic lane image (left), and high-level annotation through lane detection module (right) .................................................................................................................. 17
Figure 7 - Lane detection overlapping criterion examples. Ground truth manual annotation (blue), True Positive detection (green), False Positive detection (red) .................................................................................................................. 18
Figure 8: Sample of generated FP detection due to restrictions in annotations. Lane detection algorithm correctly detects a lane to the left of the ego lane (left image), but such lane is not annotated (right image). Therefore, a false positive detection is triggered during evaluation ........................................ 22
Figure 1: Handling of steep curves .................................................................................... 25
Figure 1: Handling of lane change manoeuvres ................................................................. 25
Figure 1: Handling of non-lane markings .......................................................................... 26
Figure 1: Handling of multiple lanes ................................................................................ 26
Figure 1: Behaviour in tunnels .......................................................................................... 27
Figure 9: Traffic Sign Recognition system’s general architecture .................................... 28
Figure 10: Traffic Sign Detection module architecture ...................................................... 30
Figure 11: Sliding window sampling strategy ..................................................................... 31
Figure 12: HOG Descriptor generation ............................................................................. 31
Figure 13: SVM binary classifier ...................................................................................... 32
Figure 14: Grouping of detections. Multiple hypotheses over the same traffic sign (left) are grouped into a single final hypothesis (right) ........................................................................................................ 32
Figure 15: Example creation of a clutter track (in orange), and its consolidation as a track after a number of consecutive detections have been associated to it ................................................................. 34
Figure 16: Robust optical flow computation for the last track position. Example using person detection .................................................................................................................................................................. 35
Figure 17: AlexNet neural network architecture ................................................................. 36
Figure 18: Data augmentation process for training purposes .............................................. 37
Figure 19: Fitting traffic sign in image patches ................................................................... 37
Figure 20: Classification pipeline ..................................................................................... 38
Figure 21: Tracking-classification validation strategy ......................................................... 39
Figure 22: Simplified representation of a point P in world coordinates projected into the image plane .................................................................................................................................................. 40
Figure 23: Typical lenses distortions ................................................................................ 41
Figure 24: Calibration pattern (asymmetric circle grid pattern) ......................................... 42
Figure 25: Homography analytical equation ..................................................................... 43
Figure 26: Analytical expression of distortion correction model and pixel coordinates conversion model .................................................................................................................................................. 43
Figure 27: Assumptions to estimate the location of a traffic sign ........................................ 44
Figure 28: Image Feature Tracking in LibVISO2 ................................................................. 48
Figure 29: Motion hypothesis space. Each black dot is a RANSAC hypothesis and the magenta circle is the best scoring hypothesis. The covariance matrix with respect to the best scoring hypothesis is also visualized .................................................................................................................................................. 49
Figure 30: Screenshot of TUE calibration engine that allows calibration (stereo) cameras on thousands of images automatically .................................................................................................................. 50
Figure 31: Visual Odometry multi-threading pipe-line ........................................................ 51
Figure 32: Screenshot of Visual Odometry engine in RTMAPS .......................................... 52
Figure 33: Results on several stereo datasets. The left columns show the GPS ground truth; the middle columns show the visual odometry results with respect to the first frame in the sequence for which absolute pose information is available; the right columns show the results after visual map matching to reduce error build-up. Quantitative results are provided in Table 8 ........................................... 54
Figure 34: Results of Visual Odometry with visual map matching compared to GPS .......... 55
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Evaluated video subsets for lane detection algorithm.</td>
<td>16</td>
</tr>
<tr>
<td>Table 2</td>
<td>Lane detection performance (curvature range 500 meters-infinity)</td>
<td>20</td>
</tr>
<tr>
<td>Table 3</td>
<td>Lane detection performance (curvature range 250-500 meters)</td>
<td>20</td>
</tr>
<tr>
<td>Table 4</td>
<td>Lane detection performance (curvature range 50-250 meters)</td>
<td>20</td>
</tr>
<tr>
<td>Table 5</td>
<td>Lane detection performance (curvature range 30-60 meters)</td>
<td>21</td>
</tr>
<tr>
<td>Table 6</td>
<td>False Negative Rate for ego lane</td>
<td>22</td>
</tr>
<tr>
<td>Table 7</td>
<td>TRL levels of visual odometry engine throughout project</td>
<td>51</td>
</tr>
<tr>
<td>Table 8</td>
<td>Visual odometry results on 5 datasets</td>
<td>55</td>
</tr>
</tbody>
</table>
1. Executive Summary

This document is the public deliverable D2.3 of the H2020 project entitled *Low Cost GNSS and Computer Vision Fusion for Accurate Lane Level Navigation and Enhanced Automatic Map Generation*, denoted by INLANE. This document reflects the work done during the first year of the project in three tasks of the Work Package 2 (GNSS and Vision based road and scene element geo-localisation and characterization): T2.2 Design and development of vision-based lane-level road parameter estimation, T2.3 Design and development of vision-based traffic sign detection and recognition algorithms, and T2.4 Design and development of vision-based relative localisation algorithms and map matching algorithms for absolute localisation.

The document is organised in three main sections, one for each of the software modules that are described. Each main section presents the design, implementation and results of the corresponding software module.
2. Introduction

2.1 Purpose of Document

The aim of this document is to describe the vision-based software modules developed during the first year of the INLANE project. These software modules correspond to the first prototype of INLANE, so the work is still in progress and the modules lack the complete functionality and TRL level of the planned final prototype.

This document is devised to be a working document that gets refined to reflect new developments carried out during the lifetime of the project. The final versions of the vision-based software modules will be reported in D2.4 (Report on developed vision-based software modules v2) in the 30th month of the project.

2.2 Intended Audience

The dissemination level of D2.3 is public. The intended audience are the GSA, the consortium members and the whole Computer Vision community.
3. Lane detection

This section describes the work carried out to design, implement and evaluate a lane detection algorithm, which uses as input monocular images.

This method is designed to work well in highway scenarios, under the hypothesis that lane markings exist and are visible.

A calibration file is required, which determines the intrinsic parameters of the camera, and the relative position and rotation of the camera with respect to the world coordinate system (to be placed in the road plane). The calibration can be loaded from an external process, or automatically estimated via a vanishing point detection algorithm.

This algorithm is able to detect lane markings in the scene, and to fit a multi-lane model, where each lane is modelled as a second order curvature in the road-plane.

The implementation of the algorithm is mature and optimized (a C++ version of the algorithm is available inside the Viulib library by Vicomtech-IK4). Preliminary evaluation activities have shown excellent results of the algorithm under the defined assumptions, and very high efficiency in terms of computational load.

3.1 Design

The lane detection algorithm implements a method to detect lane markings in image sequences, to fit a curvilinear model to them, and to obtain measurements from the model, such as the geometry, width, curvature, etc.

The basic assumption is that lanes are entities delimited by lane markings, which are bright stripes along the road. Detection of lane markings is a critical first step. In the absence of lane markings (or bad painted or not visible), the lane detection pipeline cannot provide valid output. A confidence analysis allows the algorithm to identify such situations, and therefore to switch off and provide no incorrect output.

Figure 1 shows the block diagram of the proposed algorithm:
Calibration

The lane detection is based on the assumption that the road is flat in the local surroundings of the camera. This hypothesis holds true for most scenarios for typical camera ranges (15-30 metres).

The algorithm is prepared to load existing calibration files, or to start a vanishing point estimation thread, whose aim is to generate an automatic calibration of the scene.

After the calibration has been loaded or computed, an Inverse Perspective Mapping (IPM) setup process generates a Look-Up Table for the efficient transformation between points in the image domain and their correspondence in the road-plane.

Lane markings segmentation

The detection of lane markings is the first step of the algorithm, and is based on a bump detector, which scans rows of the input image analyzing its intensity in order to find patterns that correspond to sections of bright stripes (which correspond to lane markings).

The detection is carried out at a resized image (there is no need to compute this step at full resolution), and on selected rows, to reduce the computational complexity.

The algorithm uses a threshold value that is computed dynamically according to previous results.
The detected points are transformed into the road-plane domain, and a Connected Component Analysis (CCA) is carried out to join points into clusters or stripes.

**Lane tracking**

The detected stripes are projected into the x-axis to produce a histogram of x-positions.

This histogram shows peaks at the x-position of dominant lane markings. The local maxima of this histogram can be used to feed a linear tracking algorithm (the Kalman Filter), that updates the x-position, and the width of the observed lanes.

**Curvature analysis, multi-lane & Confidence Analysis**

The points that have been transformed with the homography can be grouped using a
Connected Component Analysis that connects points from bottom to top using a spatial similarity criteria.

![Figure 4: Example parabolic fitting in: (left) a straight road; (right) a curvy road.](image)

Each of the connected groups are called “stripes”, and a Parabola can be fitted using least squares method. As a result, the curvature parameters of the road can be measured for each image, and fed into a Kalman Filter that provides the required temporal consistence.

![Figure 5: Kalman filtered road model: in white, the detected points, in blue, the lane model predicted with the Kalman Filter: (left) straight road; (right) curvy road.](image)

The result of the Kalman Filter is then used to update the information of the multi-lane model, which contains all the relevant information about the roads: number, position, curvature, etc of the lanes. A confidence analysis is carried out to assess the feasibility of the obtained results, and to ensure that reasonable values are provided.

### 3.2 Implementation

The proposed algorithm is implemented and integrated into the Viulib library by Vicomtech-IK4. This library is focused on computer vision and machine learning for a number of different sectors and applications.
The lane detection algorithm is implemented under the viulib_adas module (Advanced Driver Assistance Systems).

The implementation uses the C++ programming language, and is intended to be a multi-platform library that can be used in a wide variety of Operative Systems (including Android, iOS, Windows, Linux).

The structure of the viulib_adas module includes the definition of a number of base classes, which are summarized in the following code snippet:

```cpp
namespace viulib{
    ...
    namespace adas{
        /* Containers of information */
        class LaneMarking {...};
        class Lane {...};
        class MultiLane {...};
        class RoadObject {...};
        class Vehicle : public viulib::adas::RoadObject {...};
        class Pedestrian : public viulib::adas::RoadObject {...};
        class TrafficSign : public viulib::adas::RoadObject {...};

        /* LaneDetector receives images, and keeps and update a MultiLane object. */
        class LaneDetector : public VObject {...};
        class VanishingPointDetector : public VObject{...};
        ...
        class VehicleDetector {...};
    }
}
```

The usage of the detector implies calling the desired lane detector object and configuring its parameters:

```cpp
using namespace viulib::adas;
using namespace viulib::calibration;

/* Load calibration */
Plane* plane = Plane::create();
plane->loadPlane(planeFile);
...

/* Load detector */
LaneDetector* laneDetector = LaneDetector::create(“LaneDetectorMultiLaneCurvature_v2”);
laneDetector->set(args.procSize, “procSize”);
...

/* Use detector inside image loop */
laneDetector->run( image );
...```

### 3.3 Results

This section summarizes the preliminary results obtained after the evaluation of the performance of the lane detection algorithm. An evaluation methodology has been used, which consists on the selection and annotation of some test videos, the definition of parameters to evaluate, and corresponding functions to analyze the error between ground truth and detections. The analysis is detailed for different levels of curvature of the road, in
order to provide a clearer view on the impact of different type of roads on the proposed algorithm. Also, some subjective requirements or expected error values are provided, which facilitates the discussion on the obtained results.

Additionally, computational load is analyzed with an example platform and the described implementation of the algorithm.

3.3.1 Videos evaluated

The lane detection algorithm has been evaluated with objective metrics in the subset of videos detailed in Table 1 (during the development of the module, a subjective analysis has been continuously carried out; some findings are presented at the end of this section). The videos contain regions with clear lane markings, no intersections, and flat world. The scenarios comprise straight lanes, curves, and lane changes. A total amount of 4:05 minutes of video has been annotated and evaluated.

<table>
<thead>
<tr>
<th>Type</th>
<th>Sequence</th>
<th>Frame Start</th>
<th>Frame End</th>
<th>Total frames</th>
<th>Duration (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peri-urban</td>
<td>IMG_0005</td>
<td>1900</td>
<td>5000</td>
<td>3101</td>
<td>01:43</td>
</tr>
<tr>
<td>City</td>
<td>IMG_0007</td>
<td>0</td>
<td>1900</td>
<td>1901</td>
<td>01:03</td>
</tr>
<tr>
<td>City</td>
<td>IMG_0011</td>
<td>2465</td>
<td>2965</td>
<td>501</td>
<td>00:16</td>
</tr>
</tbody>
</table>

3.3.2 Performance evaluation

To evaluate the performance of the lane detection algorithm, a number of metrics have been defined, which include:

- **Detection performance**: ability to correctly identify the existence of lanes in the images, in terms of False Positive, False Negative, True Positive.

- **Lane width**: ability to provide a measure of the width of the lane, in metric units.

- **Lane curvature**: ability to provide a measure of the curvature of the lane, in terms of curvature radius.

- **Distance to lane boundaries**: these measures correspond to the metric distance between the center of the car to the lateral lane markings.

- **Relative angle to lane boundaries**: these are derived magnitudes that can be used to identify vehicle’s orientation inside the lane.

The lane ground truth information is obtained in a semi-automatic two-step procedure. First, lane markings are manually annotated using Vicomtech annotation tool. Second, a synthetic image is generated and used to compute high-level information, as explained in the next subsections.

**Lane markings annotations**

Vicomtech annotation tool is used to annotate visible lane markings in each video frame, by
drawing the polyline that models the centreline of the marking (Figure 6).

Figure 6 - Lane markings manual annotation with Vicomtech annotation tool.

High-level lane model annotation

Previously annotated lane markings are used to generate a synthetic black and white video which contains a clear view of the lanes in each frame (Figure 7). This synthetic video is then fed as input to the lane detection application in order to obtain high-level lane information: number of lanes, ego lane, width, curvature, etc.

Proper camera calibration is required to correctly generate the high-level lane annotations. Both camera intrinsic and extrinsic parameters are found following standard calibration procedures (checkerboard method for the first parameters, 4-point method for the second ones). Calibration validity is checked by comparing annotations output with map GPS information (e.g. generated lane width annotation is compared to GPS data to ensure it matches the real scenario).
This semi-automatic procedure allows generating ground truth data in a very robust and efficient way.

To evaluate the detection performance, an overlapping criterion is used, by defining a numerical range which describes both the width and position of the lane at the image bottom (the nearest visible part of the lane with respect to the on-board camera). In particular, the range is defined by the horizontal position of the lane left-most and right-most pixels (see Figure 8).

![Figure 8 - Lane detection overlapping criterion examples. Ground truth manual annotation (blue), True Positive detection (green), False Positive detection (red).](image)

Evaluation results for the lane detection algorithm are shown below for all the evaluated video subsets. For each curvature range, mean results are computed for a fair comparison.
### Table 2 - Lane detection performance (curvature range 500 meters-infinity)

<table>
<thead>
<tr>
<th>VIDEO</th>
<th>RANGE Curvature (m)</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>FNR</th>
<th>FDR</th>
<th>Width (m) Mean</th>
<th>Std</th>
<th>Curvature (1/m) Mean</th>
<th>Std</th>
<th>Distance left edge Mean</th>
<th>Std</th>
<th>Distance right edge Mean</th>
<th>Std</th>
<th>Angle left edge Mean</th>
<th>Std</th>
<th>Angle right edge Mean</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Requirement</td>
<td>500-Infinity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0%</td>
<td></td>
<td>1.0%</td>
<td></td>
<td>0.2</td>
<td></td>
<td>0.05</td>
<td></td>
<td>0.002</td>
<td></td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>IMG_0005, 1900-5000</td>
<td>500-Infinity</td>
<td>2556</td>
<td>126</td>
<td>98</td>
<td>3.7%</td>
<td>4.7%</td>
<td>0.1109</td>
<td>0.1504</td>
<td>0.0004</td>
<td>0.0007</td>
<td>0.0293</td>
<td>0.0849</td>
<td>0.0970</td>
<td>0.1302</td>
<td>0.2483</td>
<td>0.7727</td>
<td>0.4375</td>
<td>0.5009</td>
</tr>
<tr>
<td>IMG_0007, 0000-1900</td>
<td>500-Infinity</td>
<td>2748</td>
<td>510</td>
<td>73</td>
<td>2.6%</td>
<td>15.7%</td>
<td>0.0963</td>
<td>0.1706</td>
<td>0.0003</td>
<td>0.0007</td>
<td>0.0479</td>
<td>0.0960</td>
<td>0.0704</td>
<td>0.1577</td>
<td>0.2426</td>
<td>0.5534</td>
<td>0.2726</td>
<td>0.5545</td>
</tr>
<tr>
<td>IMG_0007, 2435-3175</td>
<td>500-Infinity</td>
<td>390</td>
<td>16</td>
<td>0</td>
<td>0.0%</td>
<td>3.9%</td>
<td>0.0531</td>
<td>0.0620</td>
<td>0.0004</td>
<td>0.0007</td>
<td>0.0130</td>
<td>0.0221</td>
<td>0.0505</td>
<td>0.0653</td>
<td>0.1311</td>
<td>0.1568</td>
<td>0.5976</td>
<td>1.6709</td>
</tr>
<tr>
<td>IMG_0007, 4210-4797</td>
<td>500-Infinity</td>
<td>1037</td>
<td>0</td>
<td>59</td>
<td>5.4%</td>
<td>0.0%</td>
<td>0.0969</td>
<td>0.1286</td>
<td>0.0003</td>
<td>0.0006</td>
<td>0.0402</td>
<td>0.0803</td>
<td>0.0689</td>
<td>0.1135</td>
<td>0.3652</td>
<td>0.7979</td>
<td>0.3008</td>
<td>0.3489</td>
</tr>
<tr>
<td>IMG_0011, 2465-2965</td>
<td>500-Infinity</td>
<td>893</td>
<td>31</td>
<td>8</td>
<td>0.9%</td>
<td>3.4%</td>
<td>0.0715</td>
<td>0.1361</td>
<td>0.0002</td>
<td>0.0004</td>
<td>0.0420</td>
<td>0.1114</td>
<td>0.0465</td>
<td>0.0960</td>
<td>0.1438</td>
<td>0.2966</td>
<td>0.2137</td>
<td>0.4806</td>
</tr>
<tr>
<td>IMG_0011, 5370-5825</td>
<td>500-Infinity</td>
<td>565</td>
<td>0</td>
<td>8</td>
<td>1.4%</td>
<td>0.0%</td>
<td>0.0426</td>
<td>0.0683</td>
<td>0.0002</td>
<td>0.0005</td>
<td>0.0179</td>
<td>0.0352</td>
<td>0.0322</td>
<td>0.0610</td>
<td>0.1511</td>
<td>0.1888</td>
<td>0.2296</td>
<td>0.4761</td>
</tr>
<tr>
<td>Mean results</td>
<td>500-Infinity</td>
<td>8189</td>
<td>683</td>
<td>246</td>
<td>2.9%</td>
<td>7.7%</td>
<td>0.0927</td>
<td>0.1431</td>
<td>0.0003</td>
<td>0.0006</td>
<td>0.0368</td>
<td>0.0846</td>
<td>0.0725</td>
<td>0.1259</td>
<td>0.2384</td>
<td>0.5837</td>
<td>0.3339</td>
<td>0.5493</td>
</tr>
</tbody>
</table>

### Table 3 - Lane detection performance (curvature range 250-500 meters)

<table>
<thead>
<tr>
<th>VIDEO</th>
<th>RANGE Curvature (m)</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>FNR</th>
<th>FDR</th>
<th>Width (m) Mean</th>
<th>Std</th>
<th>Curvature (1/m) Mean</th>
<th>Std</th>
<th>Distance left edge Mean</th>
<th>Std</th>
<th>Distance right edge Mean</th>
<th>Std</th>
<th>Angle left edge Mean</th>
<th>Std</th>
<th>Angle right edge Mean</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Requirement</td>
<td>250-500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.0%</td>
<td></td>
<td>5.0%</td>
<td></td>
<td>0.3</td>
<td></td>
<td>0.05</td>
<td></td>
<td>0.004</td>
<td></td>
<td>0.002</td>
<td></td>
</tr>
<tr>
<td>IMG_0005, 1900-5000</td>
<td>250-500</td>
<td>1620</td>
<td>119</td>
<td>115</td>
<td>6.6%</td>
<td>6.8%</td>
<td>0.0788</td>
<td>0.1324</td>
<td>0.0005</td>
<td>0.0007</td>
<td>0.0258</td>
<td>0.0406</td>
<td>0.0671</td>
<td>0.1249</td>
<td>0.1372</td>
<td>0.2029</td>
<td>0.3405</td>
<td>0.6088</td>
</tr>
<tr>
<td>IMG_0007, 0000-1900</td>
<td>250-500</td>
<td>115</td>
<td>107</td>
<td>4</td>
<td>3.4%</td>
<td>48.2%</td>
<td>0.0954</td>
<td>0.1377</td>
<td>0.0010</td>
<td>0.0015</td>
<td>0.0823</td>
<td>0.1388</td>
<td>0.0317</td>
<td>0.0835</td>
<td>0.5785</td>
<td>0.9016</td>
<td>0.1557</td>
<td>0.1939</td>
</tr>
<tr>
<td>IMG_0007, 2435-3175</td>
<td>250-500</td>
<td>263</td>
<td>14</td>
<td>0</td>
<td>0.0%</td>
<td>5.1%</td>
<td>0.0954</td>
<td>0.0934</td>
<td>0.0016</td>
<td>0.0011</td>
<td>0.0304</td>
<td>0.0441</td>
<td>0.0801</td>
<td>0.0841</td>
<td>0.1635</td>
<td>0.2290</td>
<td>0.4255</td>
<td>0.5008</td>
</tr>
<tr>
<td>IMG_0011, 2465-2965</td>
<td>250-500</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.4888</td>
<td>0.2666</td>
<td>0.0022</td>
<td>0.0008</td>
<td>0.0612</td>
<td>0.0404</td>
<td>0.4407</td>
<td>0.2426</td>
<td>0.1917</td>
<td>0.1564</td>
<td>2.2251</td>
<td>1.0176</td>
</tr>
<tr>
<td>IMG_0011, 5370-5825</td>
<td>250-500</td>
<td>43</td>
<td>0</td>
<td>22</td>
<td>33.8%</td>
<td>0.0%</td>
<td>0.3089</td>
<td>0.3397</td>
<td>0.0017</td>
<td>0.0012</td>
<td>0.3559</td>
<td>0.4246</td>
<td>0.2415</td>
<td>0.3349</td>
<td>1.2972</td>
<td>2.9838</td>
<td>1.4818</td>
<td>3.0027</td>
</tr>
<tr>
<td>Mean results</td>
<td>250-500</td>
<td>2058</td>
<td>240</td>
<td>141</td>
<td>6.4%</td>
<td>10.4%</td>
<td>0.0916</td>
<td>0.1352</td>
<td>0.0007</td>
<td>0.0008</td>
<td>0.0394</td>
<td>0.0577</td>
<td>0.0748</td>
<td>0.1249</td>
<td>0.1989</td>
<td>0.3256</td>
<td>0.3890</td>
<td>0.6474</td>
</tr>
</tbody>
</table>

### Table 4 - Lane detection performance (curvature range 50-250 meters)

<table>
<thead>
<tr>
<th>VIDEO</th>
<th>RANGE Curvature (m)</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>FNR</th>
<th>FDR</th>
<th>Width (m) Mean</th>
<th>Std</th>
<th>Curvature (1/m) Mean</th>
<th>Std</th>
<th>Distance left edge Mean</th>
<th>Std</th>
<th>Distance right edge Mean</th>
<th>Std</th>
<th>Angle left edge Mean</th>
<th>Std</th>
<th>Angle right edge Mean</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Requirement</td>
<td>50-250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.0%</td>
<td></td>
<td>5.0%</td>
<td></td>
<td>0.3</td>
<td></td>
<td>0.05</td>
<td></td>
<td>0.002</td>
<td></td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>IMG_0005, 1900-5000</td>
<td>50-250</td>
<td>450</td>
<td>47</td>
<td>76</td>
<td>14.4%</td>
<td>9.5%</td>
<td>0.0634</td>
<td>0.1123</td>
<td>0.0007</td>
<td>0.0009</td>
<td>0.0369</td>
<td>0.0702</td>
<td>0.0527</td>
<td>0.0973</td>
<td>0.1518</td>
<td>0.2259</td>
<td>0.2374</td>
<td>0.4112</td>
</tr>
</tbody>
</table>
### Table 5 - Lane detection performance (curvature range 30-60 meters)

<table>
<thead>
<tr>
<th>VIDEO</th>
<th>RANGE Curvature (m)</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>FNR</th>
<th>FDR</th>
<th>Width (m) Mean</th>
<th>Std</th>
<th>Curvature (1/m) Mean</th>
<th>Std</th>
<th>Distance left edge Mean</th>
<th>Std</th>
<th>Distance right edge Mean</th>
<th>Std</th>
<th>Angle left edge Mean</th>
<th>Std</th>
<th>Angle right edge Mean</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMG_0005, 1900-5000</td>
<td>30-60</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.1058</td>
<td>0.0000</td>
<td>0.0213</td>
<td>0.0000</td>
<td>0.1041</td>
<td>0.0000</td>
<td>0.0018</td>
<td>0.0000</td>
<td>8.9264</td>
<td>0.0000</td>
<td>0.0201</td>
<td>0.0000</td>
</tr>
<tr>
<td>IMG_0007, 0000-1900</td>
<td>30-60</td>
<td>70</td>
<td>7</td>
<td>0</td>
<td>9.1%</td>
<td>0.0%</td>
<td>0.1770</td>
<td>0.2786</td>
<td>0.0017</td>
<td>0.0026</td>
<td>0.1187</td>
<td>0.2225</td>
<td>0.0928</td>
<td>0.2024</td>
<td>0.5342</td>
<td>0.8859</td>
<td>0.2868</td>
<td>0.4924</td>
</tr>
<tr>
<td>IMG_0007, 4210-4797</td>
<td>30-60</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0541</td>
<td>0.0261</td>
<td>0.0005</td>
<td>0.0001</td>
<td>0.0633</td>
<td>0.0016</td>
<td>0.0277</td>
<td>0.0092</td>
<td>9.2972</td>
<td>0.5320</td>
<td>0.1719</td>
<td>0.0584</td>
</tr>
<tr>
<td>IMG_0011, 5370-5825</td>
<td>30-60</td>
<td>35</td>
<td>0</td>
<td>1</td>
<td>2.8%</td>
<td>0.0%</td>
<td>0.1421</td>
<td>0.1895</td>
<td>0.0034</td>
<td>0.0045</td>
<td>0.0050</td>
<td>0.0204</td>
<td>0.1377</td>
<td>0.1911</td>
<td>0.8008</td>
<td>1.0334</td>
<td>1.3948</td>
<td>1.4079</td>
</tr>
<tr>
<td>Mean results</td>
<td>30-60</td>
<td>108</td>
<td>7</td>
<td>1</td>
<td>0.9%</td>
<td>6.1%</td>
<td>0.1625</td>
<td>0.2420</td>
<td>0.0024</td>
<td>0.0032</td>
<td>0.0800</td>
<td>0.1497</td>
<td>0.1056</td>
<td>0.1933</td>
<td>0.8600</td>
<td>0.9200</td>
<td>0.6482</td>
<td>0.7823</td>
</tr>
</tbody>
</table>
A detailed analysis of the results for each output signal/metric is provided in the following subsections.

**False Negative Rate (FNR) and False Discovery Rate (FDR)**

FNR and FDR values are based on the traditional confusion matrix for object detection algorithms. The overlapping ratio used as criterion for associating ground truth lanes to detected lanes is set to 0.5.

Regarding the FNR, it is close to the target in almost all cases. However, it is worth mentioning that most of the false negative detections are linked to non-detected adjacent lanes. That is, the ego lane is barely missed, as proved in Table 6, which shows FNR only for the annotated ground truth ego lanes.

<table>
<thead>
<tr>
<th>RANGE</th>
<th>FNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curvature (m)</td>
<td></td>
</tr>
<tr>
<td>500-Infinity</td>
<td>0.37%</td>
</tr>
<tr>
<td>250-500</td>
<td>0.56%</td>
</tr>
<tr>
<td>50-250</td>
<td>0.87%</td>
</tr>
<tr>
<td>30-60</td>
<td>1.30%</td>
</tr>
</tbody>
</table>

On the other hand, FDR is in general larger than expected. However, this result is closely related to the evaluation procedure: due to time restrictions and annotations/evaluation complexity, only the ego lane and at most another adjacent lane have been annotated. This causes the number of false positive detections to raise in situations in which the detection algorithm is capable of correctly detect more than 2 lanes, as the ground truth data contains 2 annotated lanes at most. This situation is depicted in Figure 9. A subjective analysis of the application output shows that the number of real false positive detections is very limited.

![Sample of generated FP detection due to restrictions in annotations](image)

Figure 9: Sample of generated FP detection due to restrictions in annotations. Lane detection algorithm correctly detects a lane to the left of the ego lane (left image), but such lane is not annotated (right image). Therefore, a false positive detection is triggered during evaluation.

**Lane width**

The mean error of the lane width clearly satisfies the requirements in all videos and
curvature ranges. In fact, the resulting average values are a 50%-65% better than expected. Therefore, the detection algorithm precisely models this lane parameter.

Regarding the standard deviation of the error, it lies around 15 cm. Even though this is a relatively small quantity, it exceeds approximately 2-3 times the expected values. This nature of the signal is the main origin of this result: the algorithm is capable of outputting a precise estimation of the lane width when the conditions are optimal, but once the confidence is too low, the width might change abruptly. A bad width adjustment might differ in 20-30 cm from the true width, and these relatively high values have a great impact in the typical mean-standard deviation computation. That is, because the signal is not well modelled by the mean-standard deviation model, and such fact should be taken into account when analysing the evaluation results.

**Lane curvature**

Lane curvature errors are almost negligible in all cases, satisfying required values with a large margin. The **mean error is a 75%-90% better than expected for all curvatures** (minimum error of 0.0003 1/m for straight lanes, and maximum of 0.0025 1/m for very close curves). The error **standard deviation has also a considerable improvement of 40-65% with respect to requirements**. Therefore, the algorithm is capable of correctly model curvature in all road scenarios.

**Distance to lane boundaries**

Similarly to the lane width, distances to both left and right edge of the lane are precisely estimated, with mean errors of less than 7 cm. That implies the clear satisfaction of requirements for every curvature range. Therefore, the position of the ego vehicle within the ego lane can be considered as a very reliable measure.

The standard deviation of the error is slightly above requirements (up to 9 cm). This additional error is relatively small, and it has a very limited impact given the accuracy and repeatability of the estimation: as stated above, the mean error is much lower with respect to requirements, and that results in having an output mean+std value less than the requirement mean+std value.

**Relative angle to lane boundaries**

The estimation of the relative angle between the ego vehicle and the lane boundaries is very precise, and for that reason the error is within limits in all cases, both in terms of mean error and standard deviation. The **mean error is stable** around 0.20-0.30 degrees for all lanes with curvature radius between infinity (straight lanes) and 50 meters (close curves), which corresponds to an **improvement with respect to requirements between 65% and 90%**. For very close curves, the mean is kept low at around 0.75 degrees. The **error standard deviation** is also relatively stable and low, **staying always below 1 degree** (even in very close curves).

### 3.3.3 Computational cost

A significant part of the development effort has been devoted to accelerate the processing time of the algorithms, with the final goal of making it as lightweight as possible, and thus fit into a wide variety of computing platforms. Several strategies have been considered for that
purpose, such as row sampling, and pre-computation of maps (Look-Up Tables).

In general terms, we can say that the algorithms are fast enough to run real-time in most standard PCs. Of particular relevance is the fact that no code optimization has been carried out, i.e. only C++ code, without GPU nor low level implementation.

As an example, we present here the profile information of the execution of the algorithms into an example sequence with 1920 x 1080 pixels, using a processing size of 640 x 360 pixels.

The platform is a desktop PC with Intel(R) Core(TM)2 Quad CPU Q8300 @2.50 GHz, 4 GB RAM, Windows 7 64 bits.

Example profile output:

<table>
<thead>
<tr>
<th>Block name</th>
<th>Time (ms)</th>
<th>Num. executions</th>
</tr>
</thead>
<tbody>
<tr>
<td>*LD_APP_PREPARE</td>
<td>15.411</td>
<td>1000</td>
</tr>
<tr>
<td>*LD</td>
<td>0.772</td>
<td>1000</td>
</tr>
<tr>
<td>*LD_PREPARE</td>
<td>0.001</td>
<td>1000</td>
</tr>
<tr>
<td>*LD_SRF_ENH</td>
<td>0.350</td>
<td>1000</td>
</tr>
<tr>
<td>*LD_IPM</td>
<td>0.039</td>
<td>1000</td>
</tr>
<tr>
<td>*LD_CCA</td>
<td>0.059</td>
<td>1000</td>
</tr>
<tr>
<td>*LD_VP</td>
<td>0.001</td>
<td>1000</td>
</tr>
<tr>
<td>*LD_PARAM</td>
<td>0.001</td>
<td>1000</td>
</tr>
<tr>
<td>*LD_DET</td>
<td>0.043</td>
<td>1000</td>
</tr>
<tr>
<td>*LD_TRACK</td>
<td>0.046</td>
<td>1000</td>
</tr>
<tr>
<td>*LD_ASSESS_LATERAL</td>
<td>0.001</td>
<td>1000</td>
</tr>
<tr>
<td>*LD_CURVE_ACC</td>
<td>0.075</td>
<td>1000</td>
</tr>
<tr>
<td>*LD_LM_FIT</td>
<td>0.131</td>
<td>1000</td>
</tr>
<tr>
<td>*LD_UPDATE</td>
<td>0.003</td>
<td>1000</td>
</tr>
<tr>
<td>*LD_SET</td>
<td>0.005</td>
<td>1000</td>
</tr>
</tbody>
</table>

The lane detection algorithm consumes, in average, 0.77 ms\(^1\), which is well below the theoretical limit of 30-40 ms for real-time processing at 25 fps.

3.3.4 Subjective analysis

A subjective analysis of the algorithms performance is very interesting, especially when applied to public road videos not included in the evaluation dataset. Such analysis provides an estimation on the generalization of the algorithms in complex real-world scenarios. Also,

\(^1\) These numbers have been obtained disabling the process that writes video output, draws on images and writes the numerical results into an XML file.
the behaviour of the algorithms under special circumstances is observed.

The following list provides interesting subjective observations (and their associated images) about the algorithms:

- Very steep curves can be accurately handled in most cases (subject to the presence of continuous lane markings, which often exist in such situations).

![Figure 10: Handling of steep curves.](image1.png)

- Lane change manoeuvres are handled smoothly in most situations. The lane tracker is able to detect when the vehicle is moving towards one of the lateral lane markings (this is indicated with an arrow icon and bar at the bottom of the image), and accordingly shift the lane information from the ego-lane to the destination lane.

![Figure 11: Handling of lane change manoeuvres.](image2.png)

- The lane detection algorithm can correctly handle, in most cases, the presence of non-lane markings paintings such as arrows, zebra crossings, or other signals. The algorithm focuses on the lane markings, and ignores other painted elements. The tracking is correctly carried out until the lane markings disappear. In that moment, the algorithm auto-assessment switches off the lane tracking to not provide erroneous results.
The lane detection algorithm is capable of fitting multiple lanes. In most of the sequences used for testing, 3 lanes are detected correctly. More than 3 lanes are normally not visible in the images due to the low perspective. If higher perspectives were used, the system is expected to detect them without problems.

In general terms, behaviour in tunnels is worse than in open fields. However, as long as there is sufficient illumination, the algorithms are able to auto-adjust their internal parameters to continue detecting lanes.
Figure 14: Behaviour in tunnels.
4. Traffic Sign Recognition

This section describes the work carried out in the design and implementation of a Traffic Sign Recognition system. The system uses monocular images as input, and it is able to detect, track, and classify into adequate categories all the traffic signs present in the scene. Also, image localization is used to further retrieve real-world localization with respect to the capturing camera.

The implementation of the system contains several optimizations and is available as C++ algorithm inside the Viulib library by Vicomtech-IK4. Preliminary objective and subjective evaluations show promising results, achieving good detection and classification results while requiring limited computational power.

4.1 Design

The Traffic Sign Recognition system is composed by three main modules: detection, tracking, and classification (Figure 15).

![Figure 15: Traffic Sign Recognition system's general architecture](image)

The detection stage is responsible for analyzing the complete input image and generating a list of hypotheses, in the form of image patches, about the traffic signs location. These hypotheses are the input to the next system stage, the tracking module, which includes temporal coherence to the architecture and therefore introduces two main benefits: i) less number of false detections/classifications, as uncorrelated detection hypotheses are discarded, and ii) computational advantages regarding further classification of signs, as classification may be performed only once over a single consolidated track instead of having to classify every ungrouped traffic sign detection. Finally, the classification module determines the predefined traffic sign class that each detected and tracked patch belongs to (e.g. Speed Limit 50, or Roundabout).

An additional processing stage is carried out after classification to obtain real-world location of the traffic signs with respect to the capturing camera. This information, obtained based on camera calibration parameters, is essential to further align the vision-based algorithm output with digital maps.

The next sections describe in details each of the modules, and the advanced tracking-classification validation strategy that is used for improved performance.
### 4.1.1 Detection

**Introduction**  
The detection module is in charge of generating traffic sign location hypotheses in every frame of the input video stream. Each frame is thoroughly analyzed in order to determine which image patches can be considered as potential traffic signs. Therefore, each output detection is characterized with an image location (bounding box coordinates) and a confidence value relative to the likelihood of being a sign.

Traffic sign detection involves several challenges. First, the ones common to every vision-based detection scheme, such as coping with different possible sign sizes or changes in illumination conditions. Second, appearance is highly dissimilar among the several general signs categories (danger, mandatory, prohibition...). This introduces complications when obtaining a single detector capable of generalization (i.e. capable of detecting every type of traffic sign).

**Detection module architecture**  
The proposed detection module has been designed to cope with the previously mentioned detection challenges. Special attention has been given to the different traffic signs appearances, and for that reason, a multithreaded scheme has been implemented, where each thread contains a category-oriented traffic sign detector (Figure 16). The output hypotheses generated by each detection thread are merged in the final steps, and a unique list of hypotheses is sent to the further tracking module.

Each category-oriented detector is based on the same general scheme, which contains four main stages: pre-processing, high-recall detector, grouping, and high-precision detector.

- **Pre-processing**: the objective of this stage is to normalize image illumination conditions, so that illumination differences among frames are reduced and thus a more homogenous set of images are input to the following detectors. Having more similar appearances helps the detectors to be more robust.

- **High-recall detector**: a first detector is applied over the input image in the traditional multiscale sliding-window fashion. This detector extracts HOG features, and then uses them in an SVM classification problem to determine if the analyzed patch is a traffic sign (refer to next section for more details). The main characteristic of this detector is that it is trained to provide a high recall metric, i.e. to not miss any traffic sign in the scene, even if it means that more false positive detections are generated. At the same time, the used feature descriptor vector is small, so that detection runs very fast.

- **Grouping**: due to the multiscale sliding-window sampling strategy, a single object can produce several different detection hypotheses. Certain grouping strategy is applied in order to merge close coherent detections and to discard isolated false hypotheses.

- **High-performance detector**: a second detector is applied in order to verify previous hypotheses. For that reason, this detector is trained to have a high performance and robustness, even though that implies higher computational cost during prediction.
the end, it acts as a validation classifier over a limited number of image patches.

Classification-based detectors
Both detectors included in the detection module are classification-based detectors. Detection based on classification is a paradigm of detection in which a sampling strategy feeds a binary classifier with candidates that has been trained to identify some type of generic item (traffic signs, people, cars, ...).

Therefore, the general detection pipeline is the following. In the first place, a sampling algorithm is applied over the input image in order to generate image patches. In the proposed system, the popular sliding window is used. This strategy defines a window of size $C$ columns by $R$ rows, and moves it over the image by repeatedly applying a stride $S$ both horizontally and vertically (Figure 17). Each image patch defined by the sliding window is extracted and characterized by a feature vector.
HOG descriptor [1] is chosen to obtain such feature vectors (Figure 18). The process involves dividing the image patch in cells, which are further grouped into blocks. For each cell a histogram of gradient angles is constructed. All the histograms are concatenated to get the final vector.

This feature vector is the input to a binary classifier that decides whether the patch is a traffic sign or not. For the proposed system, a linear Support Vector Machine (SVM) [2] is used as classifier (Figure 19). Cross validation methodology is employed to optimize training and obtain improved results.
Although both detectors included in the detection pipeline follow this same general structure, there are certain differences:

- **High-recall detector**: it uses an exhaustive sliding window sampling strategy to avoid missing traffic signs in the image. However, it also uses a feature descriptor vector of reduced length to obtain fast processing. Most of the image patches will be rejected by this detector, although many false positive candidates might proceed to following stages.

- **High-performance detector**: there is no sliding window sampling applied to image patches retrieved from previous detector, or a very limited and computationally inexpensive sampling is done. The feature vector has a larger size than in the previous case, as it has to be descriptive enough for validating the hypothesis. Although this implies slower processing times, global stage time is low given the limited number of hypotheses to validate.

### Grouping of detections

The multiscale sliding window approach of the first high-recall detector causes the output list of hypotheses to be redundant, i.e. several hypotheses may arise from a single traffic sign in
The grouping algorithm is based on similarity of rectangles. Those output rectangles that are similar, in terms of both scale and position, are merged into a single rectangle bounding the potential traffic sign. For discarding false positive detections, a minimum number of overlapping rectangles is required to output a hypothesis. The different parameters of the grouping are set empirically, based on the configuration and trained model of the high-recall detector.

A similar grouping algorithm is used at the end of the detection module to merge hypotheses from the different detection threads.

### 4.1.2 Tracking

**Introduction**

Tracking is the stage in which intra-frame detections are analyzed through time in order to group them and create an inter-frame entity called track. In this way, a single object (e.g. a traffic sign) in the scene is represented by a single track, associated with as much detections as generated by the detectors during the time the object is in the scene.

The need for a tracking method is clear: a track can provide more information about the object itself, such as its motion, its shape or size variability through time, etc. Subsequent analysis can then be applied, such as action recognition, enhanced robustness for classification, perspective inference, etc. Nevertheless, detections provided by detectors account for a number of problems. Namely, detections tend to generate noisy, incorrect, missing, and time sparse observations of the objects under analysis. Considering these problems, we have designed a tracking method that is able to minimize the impact of the problems of detectors, providing time coherence to the observations.

The proposed tracking module follows a Data Association scheme, plus the utilization of an optical-flow based prediction method.

**Data Association Tracking**

The implemented tracking method works on a four-step fashion: predict, associate, correct, and create/destroy.

- **Predict**: a prediction mechanism is applied in order to predict the size and position of the tracks at the next time frame. The proposed Data Association Tracking algorithm is able to work with a number of options for this step, including simple linear prediction, template matching, optical flow, or more complex visual trackers (such as Multiple Instance Learning or Learn-Track-Detect).

- **Associate**: incoming detections are compared with existing tracks via comparison functions (e.g. overlapping between bounding boxes, Bhattacharyya distances, or any other suitable function) and a Data Association Matrix is created. A greedy algorithm
is applied to find the associations between detections and tracks.

- **Correct:** once the associations are done, the correct step updates the position and size of the track according to its associated detection.

- **Create/destroy:** new detections that cannot be associated with existing tracks are used to start new tracks, which can be labelled as clutter temporarily until some temporal coherency rule is satisfied (e.g. a number of matching consecutive detections happen, or a number of detections in a time window). The clutter technique is very useful to avoid the creation of tracks from false positive detections. Tracks are destroyed on a similar basis: when no new detections feed the track, it can be destroyed. Additional destroy criteria can be used (e.g. outside of bounds, impossible dynamics, low confidence from the visual tracker, etc.).

---

**Optical flow-based prediction**

Each track contains a number of keypoints or salient points, which are detected using any feature point detection (e.g. Harris corner detection, FAST, or more complex detectors like SIFT), or can also be defined following a fixed grid of positions inside the regions of the track (this is the option implemented in the proposed system). These points can be propagated to the next frame (they are “tracked”) by using a KLT tracker [3]. After the tracking process is completed, we roughly have the (sparse) optical flow of the last position of the track. The idea is to analyze the optical flow and determine the motion of the object to update the track. To simplify the computation, we have used a 2D translational model, i.e. displacement in x and y coordinates only.
Figure 22: Robust optical flow computation for the last track position. Example using person detection

Figure 22 depicts an example computation of the optical flow for a given track, in this case for pedestrian detection. As it can be seen, not all the vectors of the optical flow follow the real motion of the object. These vectors can be considered as outliers, while the vectors correctly representing the motion of the object are considered as inliers of the translational model. Then, a RANSAC procedure is used to separate inliers from outliers, using the 2D translational model as the function to compute the minimal sample set, the error cost, and the estimation function. We use the MSAC variety of RANSAC, which provides better performance with the same computational cost. MSAC helps generating robust estimations of 2D translational motion which is a simple yet effective motion model for short periods of time.

To allow the track handle large displacements within the image, the search area where the keypoints can be tracked has been set to the entire image. This helps the system to correctly find the same object even if it has incurred in a long displacement (possibly due to a video low frame rate).

Finally, for sequences with many objects, each track would need to compute the optical flow analysis. A more efficient analysis is to create a multi-region for all the tracks under study and launch the keypoint tracking procedure only once and then apply the RANSAC procedures on the keypoint matches windowed by the rectangles defined by each track. The reason is that some optical flow methods are optimized to work well with large images and therefore it is worthy to analyze the whole image rather than multiple times small images, which may suffer function call overhead.

Eventually, this tracker suffers when the detections are corrupted or contain multiple objects. In such case, the optical flow analysis would be multimodal, and there will be not a single 2D translational motion that best represent a set of inliers. Therefore, occlusions and complex interaction among objects are not solved by this tracker, although this scenario is very unlikely for traffic sign detection.

4.1.3 Classification

Introduction

The classification stage receives image patches of traffic signs candidates and determines to
which of the predefined classes they belong to, and with what probability. Obtaining the final label of a traffic sign is done through the combined tracking-classification validation strategy explained in Section 4.1.4. This section gives details about the classification procedure of a single image patch.

Classification is based on Deep Neural Networks, in particular on a Convolutional Neural Network (CNN). This kind of deep structure, based on the assembly of blocks formed by convolutions, subsampling, and non-linear operations, is very convenient for image classification tasks. More specifically, the system uses the AlexNet network [4], well-known due its good performance in the ImageNet challenge in 2012. Figure 23 depicts the model architecture.

![AlexNet neural network architecture](image)

**Figure 23: AlexNet neural network architecture**

*Training and data augmentation*

In the training procedure, each sample image containing a particular traffic sign is fed forward into the Convolutional Neural Network, which processes it at different abstraction levels. The first or lower layers capture detailed spatial information, while the deeper layers encode object-level knowledge, which is used for predicting the class of the traffic sign.

Data augmentation has been performed in order to improve the dataset as well as the robustness of the Neural Network (Figure 24). For each class, a set of ten perfectly cropped traffic signs with no background are selected, and then different affine transformations are applied to them over different background images. These transformations include three scale and rotation variations, in addition to six different translations. This data augmentation enhances the spatially invariant characteristic of the Neural Network. Moreover, histogram normalization has been applied to the images.
The Neural Network has been trained using a softmax loss layer as the objective function and it has been minimized via stochastic gradient descent. All tested models and configurations have been trained using as underlying framework Caffe [5].

**Prediction and patch fitting**

For prediction, an image patch is input into the Neural Network, and the corresponding class and probability are obtained. Apart from the different traffic signs labels (e.g. Speed Limit 50, Speed Limit 60, Danger Pedestrians Crossing, …), there is an additional “Unknown” class useful for discarding incorrect hypotheses received from the detection and tracking modules.

Regarding performance, even if the data augmentation increases the robustness of the model, in practice the patch detections that are provided to the Neural Network are far from ideal. They often have considerable noise, and are not always cropped tightly around the traffic sign. This may introduce some errors in the classification predictions, and for that reason a preprocess step is applied to the patches before being classified by the network. In this step contours in the image patch are searched in order to find the biggest component in the image, corresponding to the sign, and it is fitted in such a way that the traffic sign occupies the bounding box as much as possible. A sample can be found in Figure 25: the candidate image patch has too much visible background, while the biggest found element component (marked in red) is only a small part of the patch. Therefore, the traffic sign can be re-fitted inside the image patch. Notice that all patches are finally resized to a size of 256x256 pixels before being inputted into the Neural Network.
Figure 26 depicts the complete classification pipeline, including the pre-processing steps.

![Classification Pipeline Diagram](image)

### 4.1.4 Tracking-classification validation strategy

Traditional classification approaches in a detection-tracking-classification scheme are usually of two kinds: i) classification is performed over every single detection patch, possibly having different and conflicting predicted classes for the same track, or ii) classification is performed once per track, once it has ended. The first option is computationally expensive and has large redundancy, while the second one may provide incorrect outputs as a single prediction may not be robust enough. A new advanced combined tracking-validation strategy is proposed, which provides robustness by including enough redundancy, while at the same time targets computational cost optimization for real-time processing time. Figure 27 depicts a simplified diagram of this validation strategy.

The mechanism is controlled by the so-called TrackManager. This manager stores information of the tracks, including life status (still alive or dead) and classification status (classified once, classified several times, not-classified). Different classification priorities are defined based on these statuses and other information, and according to them the tracker manager organizes the different tracks in a queue. Every time the classification module is available (no classification in process), it asks for a track to classify, and the tracker manager provides the first one in the priority queue. The classification module then analyzes the provided track and classifies a limited number of patches belonging to it, discarding those classifications outputting low prediction confidence. The final label and confidence assigned to the track being classified are computed taking into account previous predictions.

Each track can be re-classified several times while it is alive. Re-classification involves prediction of a number of patches not previously classified, and update of output label and confidence if necessary. This recurrent scheme allows online adaptive classification of the different tracks: if there are few tracks being detected and the classifier is available most of
the time, tracks are re-classified for a more robust final prediction; if there are many tracks and the classifier cannot handle exhaustive classification, the priority queue ensures that every track is classified in a timely manner and with adequate prediction performance.

Figure 27: Tracking-classification validation strategy
4.1.5 Real-world location of classified traffic signs

The classified traffic signs have been detected and tracked in terms of 2D image coordinates. However, alternative metrics are needed to precisely locate the traffic signs in the real 3D world. Converting from 2D image coordinates to 3D world coordinates requires some camera calibration procedures, which are explained in this section.

Introduction to calibration

Calibration aims to obtain a mathematical model able to project any 3D real world point into the 2D camera image plane (Figure 28). Two groups of parameters are estimated: intrinsic parameters, which describe the internal camera parameters and distortion, and extrinsic parameters, which describe the model between the image coordinate system and the world coordinate system.

![Figure 28: Simplified representation of a point P in world coordinates projected into the image plane](image)

- Distortion and intrinsic parameters

Due to internal deformities, lenses usually introduce certain distortions to the captured images. The most typical distortions are the so-called barrel distortion and pincushion distortion (Figure 29). All distortions can be present in different intensity degrees, and they are modelled as a series of nonlinear factors that multiply the ideal undistorted image coordinates.
On the other hand, every camera has some intrinsic parameters, such as the focal length and the principal point, that need to be defined. At the end, the following equations can be used to model the relationship between the image coordinates and the camera coordinates:

\[
x_d = x' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1 x' y' + p_2 (r^2 + 2x'^2)
\]
\[
y_d = y' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_1 (r + 2y'^2) + 2p_2 x' y'
\]

\[
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\begin{bmatrix}
f_x & 0 & c_x \\
0 & f_y & c_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x_d \\
y_d \\
1
\end{bmatrix}
\]

where:

- \((x_d, y_d)\) is the actual image position, after radial distortion
- \(r\) is the radial distance \(\sqrt{x^2 + y^2}\) from the centre for radial distortion.
- \(k_1, k_2, k_3, k_4, k_5, k_6\) are radial distortion coefficients.
- \(p_1, p_2\) are tangential distortion coefficients
- \(c_x, c_y\) is a principal point that is usually at the image centre
- \(f_x, f_y\) are the focal lengths expressed in pixel units.

**Extrinsic parameters**

Extrinsic parameters provide the relative position of a real world point with respect to the camera location. The following equation allows the conversion between real world
coordinates system and camera coordinates system:

\[
\begin{bmatrix}
    x_c \\
    y_c \\
    z_c \\
\end{bmatrix}
= \begin{bmatrix}
    r_{0,0} & r_{0,1} & r_{0,2} & t_x \\
    r_{1,0} & r_{1,1} & r_{1,2} & t_y \\
    r_{2,0} & r_{2,1} & r_{2,2} & t_z \\
\end{bmatrix}
\begin{bmatrix}
    x_w \\
    y_w \\
    z_w \\
    1 \\
\end{bmatrix}
\]

Extrinsics Parameters

\[
x' = \frac{x_c}{z_c} \quad y' = \frac{y_c}{z_c}
\]

where:

- \((x_w, y_w, z_w)\) is the 3D point in world coordinates.
- \((x_c, y_c, z_c)\) is the 3D point in camera coordinates.
- We will denote the image coordinates of a point under ideal (non-distorted) pinhole projection by \((x', y')\), measured in units of focal-length.
- \(r_{i,j}\) are rotation values
- \(t_x, t_y, t_z\) are a displacement vector regarding to camera coordinate system

Both the intrinsic and extrinsic parameters, as well as the distortion coefficients, can be obtained through a variety of methods. The Viulib library by Vicomtech-IK4 includes several of these methods, such as the one using a pattern board with known geometry and measures.

![Figure 30: Calibration pattern (asymmetric circlegrid pattern)](image)

For calibration, multiple views of this planar object are inputted to the system. Then, all the required parameters are computed using some optimization processes. Once calibrated, all physical units and coordinate systems will be defined.
Traffic Signs location computation

In order to estimate the location of the classified sign with respect to the camera, four points of the sign will be used to compute the homography between the traffic sign plane and the image plane. Once the homography is known, the location can be estimated assuming that the dimensions of the traffic sign are known.

In computer vision, a planar homography is defined as a projective mapping from one plane to another. Thus, the mapping of points in a two-dimensional planar surface to the image of our camera is an example of planar homography. It is possible to formulate this mapping in terms of matrix multiplication if we use homogeneous coordinates to express both the viewed point and the point on the image to which is mapped:

\[
\begin{bmatrix}
    x \\
y \\
1
\end{bmatrix}
= sH
\begin{bmatrix}
x_w \\
y_w \\
z_w
\end{bmatrix}
\]

Figure 31: Homography analytical equation

where \( s \) is an arbitrary scale factor (intended to make explicit that the homography is defined only up to that factor). Using geometry and matrix algebra, we can solve this equation and retrieve the transformation matrix \( H \). The most important observation is that \( H \) has two parts: the physical transformation, which essentially locates the object plane we are viewing; and the projection \( sM \), which introduces the camera intrinsics matrix:

\[
\begin{bmatrix}
    x \\
y \\
1
\end{bmatrix}
= sM
\begin{bmatrix}
r_{0,0} & r_{0,1} & r_{0,2} & t_x \\
r_{1,0} & r_{1,1} & r_{1,2} & t_y \\
r_{2,0} & r_{2,1} & r_{2,2} & t_z
\end{bmatrix}
\begin{bmatrix}
x_w \\
y_w \\
0 \\
1
\end{bmatrix}
\]

Figure 32: Analytical expression of distortion correction model and pixel coordinates conversion model

Note that \( z_w \) is 0 because all the points of the pattern are coplanar.

The physical transformation part is the sum of the effects of some rotation \( R \) and some translation \( t \) that relate the plane we are viewing to the image plane.

As can be extracted from previous equations, \( H \) is going to be a 3-by-3 matrix. Thus, both the individual translations and rotations are computed for each view as well as the intrinsics (which are the same for all views). As we have discussed, rotation is described by three angles and translation is defined by three offsets; hence there are six unknowns for each view.

So, a known planar object (such as the asymmetric circlegrid pattern) provides eight equations, that is, the mapping of a square into a quadrilateral can be described by four \((x, y)\) points. Each new frame gives us eight equations at the cost of six new extrinsic
unknowns, so given enough images we should be able to compute any number of intrinsic unknowns.

Some assumptions will be done here in order to know location of the traffic sign with respect to the camera:
- the four square points in a detection will be coplanar
- real measures of the sign will be known

With such four points, an estimation of the traffic sign position will be computed.

### 4.2 Implementation

The proposed system is implemented and integrated into the Viulib library by Vicomtech-IK4. In particular, the Traffic Sign Recognition (TSR) algorithm is included in the Advanced Driver Assistance System module (viulibadas). The viulib_deep module is extensively used in the background for the classification tasks. This module uses the Caffe deep learning framework in its core.

All the implementation uses C++ as programming language, as it can be used on multiple different platforms (Windows, Linux, Android, iOS) and it is ideal when optimizing performance and computational cost.

The following code snippets summarize the main structures involved in the Traffic Sign Recognition system, and the main calls which allow the creation and configuration of the system:

```cpp
namespace viulib{
    ...
    namespace adas{
        /* Containers of information */
        class RoadObject { ...};
        class TrafficSign : public viulib::adas::RoadObject { ...};
        ...
        /* TSR receives images, and outputs TrafficSigns objects */
    }
}
```
The implementation includes several optimizations in each of the modules. Some of them are the following:

- **Detection module**: pre-computation of HOG features, multi-threaded execution for multiple class object detection.
- **Tracking module**: multi-ROI (Region Of Interest) analysis for simultaneous multiple object tracking, pre-computation of optical flow pyramids, set-up of sparse grid for fast optical flow computation.
- **Classification module**: separate dedicated thread for classification tasks, GPU optimization.

### 4.3 Results

This sections describes some preliminary results obtained from objective and subjective evaluation of the traffic sign recognizer algorithm. Notice that as this algorithm is still under development and optimization, no exhaustive evaluation is available yet.

#### 4.3.1 Performance accuracy

The accuracy of the system can be measured at different levels, either taking into account one or several stages, or the complete pipeline. In this regard, objective analyses have been only carried out for the classification stage, while the detection and tracking stages have been assessed subjectively. In the future, objective evaluation will be done for these stages too, by using corresponding ground truth data.
For the classification stage, evaluation metrics have been obtained using the popular German Traffic Signs Recognition Benchmark Database (GTSRB) [6]. For the moment, a subset of 17 signs categories has been used to train and test the classification model, namely the following:

- Cars no overtake allowed
- Children
- Mandatory direction straight
- No entry
- Pass by here
- Road closed
- Roundabout
- Speed limit 20, 30, 40, 50, 60, 70, 80, 100, 120
- Stop
- Yield

Having into account such categories, the trained model achieves ~ 90% of accuracy over the GTSRB test set. In the future, the model will include all the possible traffic signs categories.

Regarding subjective analysis of the classification stage, it has been observed that while certain traffic sign categories are clearly discernible between each other, there are others more similar that affect accuracy of the system. For that reason, developing a hierarchy of classifiers is being considered for future implementations in order to increase performance.

About detection and tracking modules, subjective evaluation carried out over a variety of videos and scenarios shows promising results, as good detection rates are obtained while maintaining the computational load within reasonable limits. Some advanced techniques are under development for reducing the number of false positives, and for maximizing the accuracy of the image patch location (best possible fit around the traffic sign).

The combined general system performance has also been assessed subjectively to the date, and satisfactory results have been observed. The adaptive tracking-classification methodology reports good outputs, optimizing running times and computational load. Traffic signs are detected in nearly all cases, with accurate associated classifications, and almost no false positives traffic signs are obtained in the final stage.

### 4.3.2 Computational load

Computational load is one of the main issues that have been taking account while developing the system, and for that reason a considerable effort has been dedicated to minimizing this term. As previously mentioned in other sections, several optimizations have been applied to the different stages; among others, multithreaded schemes and the use of GPU power.

The following table summarizes the employed time by the global system and by each of its stages on an Intel Core i5-4590 CPU 3.30GHz 8 GB RAM with a NVidia GeForce GTX 960.
card, using as input a video of length 6663 frames containing several traffic signs:

As it can be observed, processing times are reasonable, reaching real-time execution. Classification is a consuming task, but due to the threading scheme and the tracking-classification strategy, computational efficiency is achieved.

Further optimizations will be implemented, always looking for the best trade-off between performance accuracy and computational cost.

<table>
<thead>
<tr>
<th>Segment name</th>
<th>Average time</th>
<th>Nb of meas</th>
<th>First called</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor::run</td>
<td>23.0605 ms</td>
<td>6663 frames</td>
<td>4.51479e+008 ms</td>
</tr>
<tr>
<td>Detection</td>
<td>21.0141 ms</td>
<td>6663 frames</td>
<td>4.51479e+008 ms</td>
</tr>
<tr>
<td>Tracking</td>
<td>0.436046 ms</td>
<td>6663 frames</td>
<td>4.51479e+008 ms</td>
</tr>
<tr>
<td>Track_management</td>
<td>0.186492 ms</td>
<td>6663 frames</td>
<td>4.51479e+008 ms</td>
</tr>
<tr>
<td>Classification_one</td>
<td>7.8965 ms</td>
<td>268 frames</td>
<td>4.51483e+008 ms</td>
</tr>
<tr>
<td>HistogramBGR</td>
<td>0.0450319 ms</td>
<td>279 frames</td>
<td>4.51483e+008 ms</td>
</tr>
</tbody>
</table>
5. Visual Odometry

The principle of visual odometry is based on image feature tracking. This is illustrated in Figure 34. By tracking image features, their movement in the image can be determined. From these in-image movements, the 6 degree-of-freedom (relative) motion of the camera in the world (and hence also that of the vehicle) can be estimated using techniques from multi-view geometry. The visual odometry technique used in INLANE is based on LibVISO2 [8] and further improved and modified by TU/e and TomTom. The design mainly follows the work in [9,10] and the key improvements are:

- Advanced automatic key-framing to improve accuracy and stability.
- Fully multi-threaded pipeline to improve throughput (frames per second).
- Modular software architecture compatible with RTMAPS.
- Dead reckoning and delay compensation.

![Figure 34: Image Feature Tracking in LibVISO2.](image)

**RAndom Sampling Consensus (RANSAC):** At the core, the visual odometry engine uses the RANSAC paradigm [7]. In this approach, a minimal set of tracked image features is randomly selected and a motion is estimated on this minimal set. This motion is a hypothesis for the true motion of the camera (vehicle). This process is repeated several times, resulting in multiple hypotheses (typically in the order of a few hundred). Next or in parallel, each hypothesis receives a score based on how well it is supported by all tracked image features. The final output is then that single hypothesis that receives the highest score as in [7,8] or a weighted average over equally good scoring hypotheses as in [9,10]. As a refinement, we can use the selected motion hypothesis to obtain all tracked image points that correspond to its motion (called inliers) and discard points that do not correspond to its motion (called outliers). A refined motion estimate can then be obtained by estimating the motion from all inliers. For more detail on these methods we refer to [8,9,10]. A picture of a motion hypothesis space is shown Figure 35. Since [7], many different RANSAC approaches have emerged. Most notably for the case of visual odometry is preemptive RANSAC, Nister [11-13], which allows evaluating a larger number of minimal-set motion estimates in the same amount of computation time. Using some sort of multi-frame optimization or multi-frame landmark tracking is also frequently used. In the extreme case, this leads to simultaneous localization and mapping (SLAM) approaches or bundle adjustment approaches (of which graph-based SLAM is a particular example). Here, the focus is on frame-to-frame approaches only.
Relative versus absolute positioning: It is important to consider that visual odometry is a relative positioning technology. Typically, the positioning is with respect to the first frame in the sequence and for this first frame the absolute position and orientation is obtained using GNSS. The errors of visual odometry are orders of magnitudes lower than regular odometry on basis of wheel rotations and steering angle but of similar accuracy as odometry on basis of wheel rotations and yaw-rate sensors. The benefit of visual odometry is that it is a key processing step in many visual recognition tasks, such as determining the movements of independent moving objects and visual map-based positioning. In order to use visual odometry for automotive applications, it must be fused with absolute positioning systems, these can be GNSS or map-based positioning systems.

Main sources of error: Empirical results show that the error distribution in the tracked absolute pose has small variance with a significant non-zero mean, i.e. the errors are biased (note that this differs from the statistical definition of estimator bias).

It is due to recent advances in robotics, computer vision and related fields of modern science that bias is left as the main source of error growth. In earlier work the effects of bias were unobservable as VO drift was dominated by suboptimal optimization, i.e. minimizing residuals in the 3D domain instead of the image domain, and due to outlier image point correspondences. Significant increases in computational power currently allow for using methods which minimize reprojection errors and which are almost unaffected by outliers in realistic circumstances, see the references in [10]. These methods exhibit significant less error growth but their errors are still biased for trajectories that are several kilometers long. Bundle adjustment is often used as the final optimization step of many state-of-the-art visual odometers. BA minimizes a non-linear model which captures the physical and statistical relationship between scene points and their projections onto the camera plane. Such models often account for lens aberrations like radial distortion and decentering distortion. It is straightforward to verify (e.g. by using Monte-Carlo simulations) that when such models truly form the image data, then state-of-the-art VO methods exhibit zero-mean error distributions. However, results obtained from real image data are biased. This indicates that the true physical and statistical phenomena underlying real camera observations are not fully
The key sources of error of a well-designed visual odometry engine and stereo camera, are sub-pixel stereo camera calibration errors. Typically, a stereo camera is calibrated once using an extensive procedure, see Figure 36. When the stereo camera is in use, it will go through several thermal cycles. These thermal cycles will create sub-pixel changes in the camera parameters, hence the calibrated parameters no longer exactly match the true camera parameters. These small imperfections will cause small imperfections in the estimated frame-to-frame motions. By integrating all frame-to-frame motions to track the absolute vehicle pose, error will accumulate indefinitely. On-line re-calibration methods currently lack accuracy to remove sub-pixel imperfections in calibrated parameters, hence the only solution is to use external sources of absolute vehicle localization (GNSS, map matching) to reduce error build up. These external information, can also be used to compensate error build up for future estimates as is demonstrated in [10]. Clearly, visual odometry requires a well exposed image with sufficient scene information. These conditions cannot be constantly guaranteed but having multiple stereo cameras (i.e. forward, backward, sideward facing) will increase the probability of well exposed images with sufficient scene information. Again fusing with other sources of information will help to bridge gaps in the visual odometry estimates. One of the goals of this project is to validate the visual odometry engine is challenging lighting and weather conditions.

Figure 36: Screenshot of TUE calibration engine that allows calibration (stereo) cameras on thousands of images automatically.

**Technology Readiness Level (TRL):** In contrast to lane and traffic sign detections, for which commercial automotive-certified solutions are available, visual odometry is currently in a lower TRL level. One of the goals of this project is to increase the TRL level of visual odometry and to bring it closer to a commercial automotive-grade product. The planning with respect to TRL levels is provided in the table below:
Table 7: TRL levels of visual odometry engine throughout project.

<table>
<thead>
<tr>
<th>Phase</th>
<th>TRL level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before project</td>
<td>TRL 4</td>
<td>Technology validated in lab</td>
</tr>
<tr>
<td>Year 1</td>
<td>Towards TRL 5</td>
<td>Main goal is to re-package the visual odometry engine into an RTMAPS module such that it can be integrated in the INLANE prototype</td>
</tr>
<tr>
<td>Year 2</td>
<td>TRL 5</td>
<td>Technology validated in relevant environment</td>
</tr>
<tr>
<td>Year 3</td>
<td>TRL 6</td>
<td>Technology demonstrated in relevant environment</td>
</tr>
</tbody>
</table>

In the first year of the project, we have improved the visual odometry engine of TU/e and re-implemented it as an RTMAPS module. It has already been integrated with other modules of TomTom in order to realize advanced map-based positioning (see Deliverable 3.1).

5.1 Design

The design of the visual odometer follows the diagram below. Each block in this diagram runs in a separate processing thread (using Intel Thread Building Blocks), whereas the original LibVISO2 code is single threaded.

![Visual Odometry multi-threading pipe-line.](image)

The output of the pipeline is a 6 degree-of-freedom (relative) motion currently provided as a 3x4 matrix (i.e. 3x3 rotation matrix and a 3x1 translation vector). From the relative motions, the pseudo absolute position can be estimated (i.e. its position with respect to its last known true absolute position). Besides the motion, the output also contains a confidence that currently consists of the ratio of inliers versus outliers, i.e. the higher the number of inliers and the lower the number of outliers, the more reliable the motion estimate is. The number of inliers can also be used (via linear error propagation) to compute a 6x6 covariance matrix expressing the uncertainty assuming a Gaussian error distribution.

5.2 Implementation

During the first year of the project, the visual odometry method is implemented as an RTMAPS module. The implementation relies on the original LibVISO2 code, on Intel Thread Building Blocks, and on OpenCV and Eigen3. Currently, the frame grabber (loader) can also be made available separately to improve modularity. An example RTMAPS solution file showing the visual odometry engine is shown below.
An example of the key code blocks required to run the visual odometry engine are provided below.

```cpp
// init visual odometry
VisualOdometryStereo::parameters *visoParams;
visoParams = new VisualOdometryStereo::parameters();

// make thread building blocks
BB2Grabber loader;
VisoTBB fextractor ( &visoParams, FEXTRACTOR);
VisoTBB matcher      ( &visoParams, MATCHER    );
VisoTBB estimator    ( &visoParams, ESTIMATOR  );
VisoTBB matchRefiner ( &visoParams, MREFINER  );
VisoTBB estimateRefiner ( &visoParams, EREFINER );

// create the image token ringbuffer
VisoTokenRingBuffer visoTokenRingBuffer( nthreads );

// init the thread building blocks
loader.initLoader( &quit, &visoTokenRingBuffer );
fextractor.initFextractor();
matcher.initMatcher();
estimator.initEstimator();
matchRefiner.initMatchRefiner();
estimateRefiner.initEstimateRefiner();
rosMessageSender.initROSMessageSender(argc,argv);

// Create the pipeline
tbb::pipeline pipeline;
```
5.3 Results

On a modern CPU (e.g. Corei7), the visual odometry runs at approximately **20 FPS with a delay of 100 ms per frame**. The error accumulation (or drift) due to the relative nature of visual odometry is approximately **5 mm and 0.5 milli degree per travelled meter**. Although, the exact results and validation within INLANE still need to be obtained and performed, we show some preliminary results in the table and figures below. The reported error metric is the average absolute difference between GPS absolute positions and visual odometry positions.
Figure 39: Results on several stereo datasets. The left columns show the GPS ground truth; the middle columns show the visual odometry results with respect to the first frame in the sequence for which absolute pose information is available; the right columns show the results after visual map matching to reduce error build-up. Quantitative results are provided in Table 8.
As mentioned earlier, the RTMAPS visual odometry module is currently used by TomTom to realize advanced map based positioning (see Deliverable 3.1). The next steps is to perform unit testing and determine its accuracy and its boundary conditions.
6. Conclusions

This report has presented the design, implementation and results of three vision-based software modules that are part of INLANE architecture:

- Lane detection
- Traffic Sign Recognition
- Visual Odometry

These modules have been developed inside the framework of WP2, which objective is to obtain a GNSS and vision based road and scene element geo localisation and characterisation. In the current version, all these modules are independent from each other and process raw video data. Their outputs feed further software modules with the aim of obtaining the desired enhanced positioning and scene modelling.

The results obtained during the first year of the project are very promising. However, further effort is needed to refine the algorithms and to implement them in a common in-vehicle hardware platform in order to conduct end-to-end field tests that would validate the complete system. This is expected to be done during the second year of the project.
7. References


[8] Andreas Geiger and Julius Ziegler and Christoph Stiller. StereoScan: Dense 3D Reconstruction in Real-time. In proceedings of IEEE Intelligent Vehicles Symposium (IV), 2011


