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

D. 4.1

Report on Prototype Integration, Validation and Testing Protocols 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>Project co-funded by the European Commission within Horizon 2020 and managed by the European GNSS Agency (GSA)</th>
<th>Dissemination level</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU Public</td>
<td>X</td>
</tr>
<tr>
<td>PP Restricted to other programme participants (including the GSA)</td>
<td></td>
</tr>
<tr>
<td>RE Restricted to a group specified by the consortium (including the GSA)</td>
<td></td>
</tr>
<tr>
<td>CO Confidential, only for members of the consortium (including the GSA)</td>
<td></td>
</tr>
</tbody>
</table>
## Document Control Sheet

<table>
<thead>
<tr>
<th>Deliverable number:</th>
<th>D4.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deliverable responsible:</td>
<td>VICOM</td>
</tr>
<tr>
<td>Work package:</td>
<td>WP4</td>
</tr>
<tr>
<td>Editor:</td>
<td>Gorka Vélez</td>
</tr>
</tbody>
</table>

## Author(s) – in alphabetical order

<table>
<thead>
<tr>
<th>Name</th>
<th>Organisation</th>
<th>E-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gijs Dubbelman</td>
<td>TUE</td>
<td><a href="mailto:g.dubbelman@tue.nl">g.dubbelman@tue.nl</a></td>
</tr>
<tr>
<td>Julian Eggert</td>
<td>HRI</td>
<td><a href="mailto:julian.eggert@honda-ri.de">julian.eggert@honda-ri.de</a></td>
</tr>
<tr>
<td>Axel Koppert</td>
<td>TCA</td>
<td><a href="mailto:axel.koppert@teleconsult-austria.at">axel.koppert@teleconsult-austria.at</a></td>
</tr>
<tr>
<td>Gorka Vélez</td>
<td>VICOM</td>
<td><a href="mailto:gvelez@vicomtech.org">gvelez@vicomtech.org</a></td>
</tr>
</tbody>
</table>

## Document Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Modifications Introduced</th>
<th>Modified by</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0.1</td>
<td>2016-09-02</td>
<td>Table of contents</td>
<td>Gorka Vélez</td>
</tr>
<tr>
<td>V0.2</td>
<td>2016-09-16</td>
<td>First version of general strategy and development, integration and testing methodology</td>
<td>Gorka Vélez</td>
</tr>
<tr>
<td>V0.3</td>
<td>2016-10-13</td>
<td>First version of E2E evaluation metrics and definition of scenario of field tests</td>
<td>Gijs Dubbelman</td>
</tr>
<tr>
<td>V0.4</td>
<td>2016-11-21</td>
<td>Modification of table of contents</td>
<td>Gorka Vélez</td>
</tr>
<tr>
<td>V0.5</td>
<td>2016-11-29</td>
<td>Revision and completion of testing strategy</td>
<td>Axel Koppert</td>
</tr>
<tr>
<td>V0.6</td>
<td>2016-12-02</td>
<td>Update of test scenarios and HW for field tests</td>
<td>Gijs Dubbelman</td>
</tr>
<tr>
<td>V0.7</td>
<td>2016-12-09</td>
<td>Year 1 testing results of Sensor Data Fusion, Sensor-to-Map alignment and Visual Odometry</td>
<td>Axel Koppert</td>
</tr>
<tr>
<td>V0.8</td>
<td>2016-12-13</td>
<td>Year 1 testing results of Lane Detector, TSR and Navigation application</td>
<td>Gorka Vélez</td>
</tr>
<tr>
<td>V0.9</td>
<td>2016-12-14</td>
<td>Abstract, Executive Summary, Introduction and Conclusions Section</td>
<td>Gorka Vélez</td>
</tr>
<tr>
<td>V0.10</td>
<td>2016-12-15</td>
<td>Corrections and format adjustments after first internal review by David Betaille</td>
<td>Gorka Vélez</td>
</tr>
<tr>
<td>V0.11</td>
<td>2016-12-16</td>
<td>Results of Local Dynamic Maps module</td>
<td>Julian Eggert</td>
</tr>
<tr>
<td>V1.0</td>
<td>2016-12-23</td>
<td>First delivered version after second review by David Betaille and Miguel Ortiz</td>
<td>Gorka Vélez</td>
</tr>
</tbody>
</table>
Abstract

This document presents the first report on prototype integration, validation and testing protocols. It describes and clarifies choices that have been made over the course of the first 12 months of project development. The procedures 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>CI</td>
<td>Continuous Integration</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>ECU</td>
<td>Electronic Control Unit</td>
</tr>
<tr>
<td>FPS</td>
<td>Frames per Second</td>
</tr>
<tr>
<td>GBPT</td>
<td>GNSS-based positioning terminal</td>
</tr>
<tr>
<td>GNSS</td>
<td>Global Navigation Satellite System</td>
</tr>
<tr>
<td>GSA</td>
<td>European GNSS Agency</td>
</tr>
<tr>
<td>GTSRB</td>
<td>German Traffic Signs Recognition Benchmark Database</td>
</tr>
<tr>
<td>H2020</td>
<td>Horizon 2020</td>
</tr>
<tr>
<td>HD</td>
<td>High Definition</td>
</tr>
<tr>
<td>HMI</td>
<td>Human-Machine Interface</td>
</tr>
<tr>
<td>IMU</td>
<td>Inertial Measurement Unit</td>
</tr>
<tr>
<td>ITS</td>
<td>Intelligent Transport Systems</td>
</tr>
<tr>
<td>NDS</td>
<td>Navigation Data Standard</td>
</tr>
<tr>
<td>PVT</td>
<td>Position, Velocity and Time</td>
</tr>
<tr>
<td>RMSE</td>
<td>Root-Mean-Square Error</td>
</tr>
<tr>
<td>SD</td>
<td>Standard Definition</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>VRU</td>
<td>Vertical Reference Unit</td>
</tr>
<tr>
<td>WP</td>
<td>Work Package</td>
</tr>
</tbody>
</table>
Table of Contents

1. Executive Summary ........................................................................................................... 8
2. Introduction .......................................................................................................................... 9
   2.1 Purpose of document ....................................................................................................... 9
   2.2 Intended audience .......................................................................................................... 9
   2.3 Related documentation ................................................................................................. 9
3. General strategy .................................................................................................................... 10
   3.1 Planned iterations ......................................................................................................... 10
   3.2 Continuous integration approach ................................................................................ 10
   3.3 Roles in design, development, integration and testing .............................................. 11
   3.4 Teams communication ............................................................................................... 12
   3.5 Process improvement and adaptation over time ....................................................... 12
4. Development ......................................................................................................................... 13
   4.1 Software development tools ....................................................................................... 13
      4.1.1 RTMaps ............................................................................................................... 13
      4.1.2 Programming languages ................................................................................... 14
   4.2 Software architecture ................................................................................................. 14
      4.2.1 Overview ............................................................................................................. 14
      4.2.2 Communication of components ........................................................................ 15
   4.3 Version control ............................................................................................................. 15
   4.4 Versioning .................................................................................................................... 16
   4.5 Documentation ........................................................................................................... 16
      4.5.1 README.md ....................................................................................................... 16
      4.5.2 Wiki .................................................................................................................... 17
   4.6 Issue management ........................................................................................................ 17
5. Software Integration and Testing ....................................................................................... 18
   5.1 Overview ....................................................................................................................... 18
   5.2 Cloud platform ............................................................................................................ 18
      5.2.1 Specifications ....................................................................................................... 18
   5.3 Integration tools ........................................................................................................... 19
      5.3.1 Docker ................................................................................................................ 19
      5.3.2 GitLab CI ............................................................................................................. 20
   5.4 Testing strategy .......................................................................................................... 21
      5.4.1 Unit tests .............................................................................................................. 21
      5.4.2 Integration tests ................................................................................................... 22
6. Performance assessment Methodology ............................................................................. 23
   6.1 Performance Assessment according to EN 16803 .................................................... 23
      6.1.1 Performance Assessment of the Positioning Component ................................... 24
      6.1.2 Performance Assessment of the Positioning-based End-to-End applications ... 24

5
6.2 Test scenarios ............................................................................................................. 25
  6.2.1 Datasets for laboratory tests .................................................................................. 25
  6.2.2 Scenario of field tests ............................................................................................ 25
6.3 Hardware for field tests ............................................................................................... 26
7. Year 1 testing results .................................................................................................... 27
  7.1 Sensor Data Fusion .................................................................................................... 27
  7.2 Sensor-to-Map Alignment ......................................................................................... 27
  7.3 Visual Odometry ........................................................................................................ 27
  7.1 Lane Detection ........................................................................................................... 27
  7.2 Traffic Sign Recognition ............................................................................................. 28
  7.3 Static map .................................................................................................................. 28
  7.4 Local Dynamic Map ................................................................................................... 28
  7.5 Navigation application ............................................................................................... 29
8. Conclusions .................................................................................................................... 30
List of Figures

Figure 1 Continuous Integration cycle .................................................................................. 11
Figure 2 RTMaps Studio ....................................................................................................... 13
Figure 3 INLANE’s software architecture .......................................................................... 14
Figure 4 INLANE’s repository in GitLab ............................................................................. 15
Figure 5 GitLab's wiki ........................................................................................................ 17
Figure 6 GitLab's issue management .................................................................................. 17
Figure 7 Integration and testing cycle ................................................................................. 18
Figure 8 Docker container in INLANE ................................................................................ 19
Figure 9 INLANE approach to install new SW dependencies ............................................. 20
Figure 10 GitLab CI architecture (figure from https://about.gitlab.com/gitlab-ci/) ........... 21
Figure 11 Unit test architecture .......................................................................................... 22
Figure 12 Architecture of an integration test step ............................................................... 23
Figure 13: Classification of the INLANE high-level subsystems according to EN16803-1 23
Figure 14 Recorded datasets (yellow, orange, blue) ............................................................ 25
Figure 15 In-vehicle hardware ........................................................................................... 26

List of Tables

Table 1: Accuracy performance features and metrics for the positioning component output .......... 24
1. Executive Summary

This document is the public deliverable D4.1 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 the work package 4 (WP4), entitled Integration and testing. More specifically, in tasks T4.1 Definition of the methodology, T4.2 Integration of prototype, and T4.3 Evaluation of prototype.

First, this document introduces its purpose, the intended audience and the related documentation. After this, the general strategy is explained, including a description of the planned iterations, the different roles among the technical staff, the channels and tools for facilitating team communication, and how the process will be improved and adapted during the duration of the project.

Then, the software development, integration and testing methodologies are explained, detailing how the proposed tools are used. Finally, the system performance assessment methodology is described and the testing results obtained in the first year are presented.
2. Introduction

2.1 Purpose of document

The main aim of this document is to describe the integration, testing and validation methodology defined during the first year of the INLANE project. Since the integration and testing stages cannot be completely decoupled from the development, the defined development tools and procedures are also described. Additionally, the testing results obtained in the first year of the project are summarised.

This document is intended to be a reference during the integration and test stages planned for each project cycle, to create consistent prototypes that will help evaluating the project expected progress against the defined goals. This document is also devised to be a working document that gets refined and updated during the lifetime of the project. The final version will be reported in D4.2 (Report on Prototype Integration, Validation and Testing Protocols v2) in the 30th month of the project.

2.2 Intended audience

This document is intended to be a handbook for internal use during the development, integration, testing and validation of INLANE’s prototypes. All people participating in these stages shall have this document as a reference.

The dissemination level of this deliverable is public. The procedures explained here can be useful for:

- System integrators.
- Software developers.
- Standardisation organisations.

2.3 Related documentation

WP4 integrates and tests the software modules developed in WP2 and WP3, following an architecture that was previously designed in WP1. Therefore, in order to understand and appreciate the work done in WP4, it is also necessary to read the deliverables generated in WP1, WP2 and WP3. More specifically:

- D1.1: to understand the requirements, the general architecture and interfaces.
- D2.1: to understand the sensor data fusion and check detailed test results.
- D2.3: to understand the vision-based software modules and check detailed test results.
- D2.5: to understand the camera-to-map data alignment and check detailed test results.
- D3.1: to understand the navigation application and check detailed test results.
3. General strategy

3.1 Planned iterations

The planned work will be developed using an iterative process (specifications, prototyping and testing). The output of each phase will contribute to the specification and design process of the incoming activities.

The work plan is scheduled into three development cycles or phases:

- Initial prototype (Alpha): The goal of this prototype is to integrate in a fast manner the existing technologies and background knowledge of the consortium in a “rough and ready” platform that can be used to scope and define the goals of end-user requirements and verify platform designs. This prototype will be tested in laboratory or in a controlled scenario. (Due to month 12).

- Second Prototype platform (Beta): The Beta version of the platform will contain the project results developed the first two years of the project. This version of the platform will be viewed as feature complete. This prototype will be tested at DITCM/TASS facilities and if possible in Barcelona. (Due to month 24).

- Final Prototype Platform: This will be the final INLANE platform prototype and will be functionally complete. The final prototype will be subjected to a second and final series of tests where it will be measured against the architecture and user requirements defined in WP2. The Final Prototype will serve as the basis for final testing and for completing project dissemination and exploitation tasks. This final prototype will be extensively tested by end users in Barcelona city. (Due to month 30).

3.2 Continuous integration approach

Traditionally, developers compile and test locally their software modules, and when they decide that they have a significant contribution, they upload it to the software versioning system. This procedure is usually followed without interruptions for several months, incrementally generating new code. The problem is that each module has grown independently, without checking the compatibility with other modules. When the integration time comes, incompatibilities and other integration problems arise. The main aim of Continuous Integration is to prevent critical integration problems, promoting the integration as part of the entire development cycle.

Continuous Integration, in its simplest form, involves a tool that monitors your version control system. If a change is detected, this tool automatically compiles and tests your application. If something goes wrong, the tool immediately notifies the corresponding developers so that they can fix the issue. This typical Continuous Integration workflow is depicted in Figure 1. The Continuous Integration tool used in INLANE is introduced in Section 5.3.2.
However, Continuous Integration is much more than a simple modification in the development cycle. Continuous Integration reduces integration risk by providing faster feedback. It is designed to help identify and fix integration and regression issues faster, resulting in a reduction on delivery time and bug number. By providing better visibility on the state of the project, it can open and facilitate communication channels between team members and encourage collaborative problem solving and process improvement. Furthermore, it automates the deployment process, reducing time to market in a reliable way.

INLANE implements a Continuous Integration methodology with the aim of obtaining the mentioned benefits. In Continuous Integration implementations, the development, integration and testing stages are tightly coupled, as one stage triggers the following. This document describes the methodology and tools used in these three stages, and the results obtained with the first prototype.

3.3 Roles in design, development, integration and testing

INLANE’s work package 4 contains the tasks related to integration and testing of the software modules designed in work package 1, and developed in work packages 2 and 3. All these work packages need to be perfectly coordinated to assure a smooth integration and testing. One of the most important points to obtain a good coordination is to define clear roles in the whole design, development, integration and testing cycle. The following technical roles have been defined in INLANE:

- Software architect. Its role is to define the overall architecture and the corresponding inter-working interfaces. There is at least one software architect per each technical partner. The corresponding tasks are done inside WP1, and are led by WP1 leader.
- Software developer. The software developer is responsible for the coding of software components. This work is performed within WPs 2 and 3.
- Unit tester. The unit tester role is to design and perform the unit tests for software components (explained in Section 5.4). Each software module has a leader that assigns a unit tester to test its module. This work is done inside WP4.
- Integrator. The integrator is the responsible for the integration of software components. This work is performed inside WP4 and is led by WP4 leader.
• System tester. Its role is to perform tests on the overall integrated system. This work is done in WP4 to assess technical performance and in WP5 to evaluate how the overall system meets users' needs.

• End user. The end user is a person who ultimately uses or is intended to ultimately use the product(s) derived from INLANE. End users are not part of the INLANE consortium but are involved in the project by the work done in WP5.

3.4 Teams communication

Projects based on distributed software development always bring communication challenges. The following tools and procedures are used in INLANE to ensure fluent communication among partners:

• Weekly conference calls, where the progress in development, integration and testing is reported and discussed. The conference calls are run using Webex, which has the additional possibility of screen sharing. All partners are invited to these progress-monitoring conference calls.

• Dedicated conference calls are also scheduled on demand. Only partners involved in the discussed issue attend the meeting.

• A mailing list dedicated to WP4 is used for sending important information to all partners involved in integration and testing.

• A chat or messaging application is included in Podio management platform. All people registered in INLANE's workspace in Podio can open private chats with other INLANE contributors.

• Podio is also used for other several communication tasks:
  o Inform partners of new scheduled meetings.
  o Confirm attendance to meetings.
  o Inform partners of new draft versions of deliverables or of any other kind of document uploaded to Podio.
  o Give brief feedback about uploaded documents.

3.5 Process improvement and adaptation over time

The process and procedures of development, integration and testing are a topic of discussion in the weekly INLANE conference calls as well as in Podio chats and e-mail conversations. If there are indications that changes in the strategy are necessary, the defined procedures will be revised in discussion with all WP1, WP2, WP3 and WP4 participants. A version of this document will be made available in Podio and changes to the procedures will be updated there and will be communicated through INLANE's mailing list.
4. Development

4.1 Software development tools

4.1.1 RTMaps

The synchronisation and fusion of real-time data streams coming from different sensors is one of the biggest technical challenges in INLANE. The automotive industry employs toolkits or middleware specifically designed to deal with this problem. RTMaps\(^1\) is one of the most used options, and the one chosen by INLANE.

RTMaps is a software platform designed to face multisensor challenges and to allow engineers and researchers to take advantage of an efficient and easy-to-use framework for fast and robust developments. It is composed of several independent modules:

- The RTMaps studio: the studio is RTMap's graphical integrated development environment (IDE). It allows assembling components, configuring them and assessing the performance.
- The RTMaps runtime engine: this is the core of any RTMaps-based application. It allows running an application developed with RTMaps using the console, without the burden of the IDE.
- The RTMaps SDK: using this SDK you can develop your own components using C/C++ or Python.

INLANE's software modules are developed using RTMaps SDK. Each module is a RTMaps component or package. The packages are assembled and tested using RTMaps Studio. Once, the complete solution is integrated and tested using RTMaps Studio, it is tested in the final embedded platform using RTMaps runtime.

\(^1\) [https://intempora.com/products/rtmaps.html](https://intempora.com/products/rtmaps.html)
4.1.2 Programming languages

Modules can be developed in any of the two programming languages compatible with RTMaps: C/C++ or Python. The modules are connected using RTMaps diagrams.

4.2 Software architecture

4.2.1 Overview

INLANE’s software architecture was designed by INLANE’s software architects in WP1 and is explained in detail in deliverable D1.1. The high-level software architecture is depicted in the following figure. For a more detailed description, please consult deliverable D1.1.

Figure 3 INLANE’s software architecture
4.2.2 Communication of components

Each software module is implemented in a RTMaps package. So all software components are arranged in a RTMaps diagram and communicated by RTMaps interfaces. However, it is still necessary to define these interfaces. This task was done in WP1 and it is explained in deliverable D1.1.

4.3 Version control

GIT is the chosen tool for version and source control. Git is a free and open source distributed version control system.

In order to ease the management and maintenance of a GIT repository, GitLab Community Edition was installed in a server. GitLab is a web-based Git repository manager with wiki and issue tracking features, using an open source licence. INLANE’s GIT repositories are private and can only be accessed by consortium members.

INLANE promotes a common style in the Git commit messages. Developers have to take into account these rules when writing the commit message:

- The message should contain a subject and a body. In case of self-explanatory subjects, the body is not necessary.
- The subject and the body are separated with a blank line.
- The subject should have a maximum of 50 characters.
- The subject text begins with a capital letter and does not end with a period.
- The imperative mood is used in the subject line. For example, “Fix bug” instead of “Fixed bug” or “Bug fixes”.
- The body should have a maximum of 72 characters.
- Bullet points are acceptable inside the body. Bullets are defined with a hyphen or asterisk, followed by a single space.
- If the commit resolves an issue registered in the issue tracker, the commit message needs to reference it.
There are many Git repositories that follow this set of rules, which is becoming a *de facto* standard in the community. An example of a well-known Git repository that follows these rules is the one from Linux kernel: https://github.com/torvalds/linux/commits/master

### 4.4 Versioning

Software versioning is the process of assigning a unique version number to a unique state of the software. INLANE uses the Semantic Versioning procedure defined in [http://semver.org/](http://semver.org/).

In a nutshell, the version number has a format of MAJOR.MINOR.PATCH, and is incremented the following way:

- **MAJOR** version when you make incompatible API changes,
- **MINOR** version when you add functionality in a backwards-compatible manner, and
- **PATCH** version when you make backwards-compatible bug fixes.

The complete specification is detailed in [http://semver.org/](http://semver.org/).

Each module has its own package versioning that it is defined in RTMaps. Each module leader is responsible of its numbering. The complete solution has another independent versioning and it is defined in the corresponding RTMaps diagram. The system integrator is the responsible of this latter numbering.

### 4.5 Documentation

#### 4.5.1 README.md

Each software module must have a README.md file placed in the root folder where the module source code is stored.

This file must be written using Markdown markup language[^2] and it must include:

- Module name.
- Brief description of module.
- Module responsible (name, organisation and e-mail address)
- Interface
  - Inputs
  - Outputs
- Dependencies (including instructions of how to install them in Ubuntu via command line)
- Test dataset(s)
  - Dataset name
  - Description of dataset
  - Path to the dataset in the repository
  - Dataset license (if applicable)
- Quick use example
- Other interesting info (if applicable)

The objective of this readme file is to contain all the necessary information for the integration and testing of the corresponding module.

4.5.2 Wiki

GitLab includes a full wiki per repository. This allows partners to document project integration and testing procedures and how-to’s. The integrator or tester can then search through previously tagged key words for relevant topic information.

Figure 5 GitLab’s wiki

4.6 Issue management

GitLab also includes an issue management system. The objective is to have a centralised tool, where partners can register errors and missing features detected during the integration and testing. The issues are first assigned to the module leader, who then assigns them to the corresponding developer.

Figure 6 GitLab’s issue management
5. Software Integration and Testing

5.1 Overview

Following the Continuous Integration approach, INLANE’s integration and testing stages are coupled. The integration is planned to be done gradually, incrementally putting together more software modules. Therefore, the testing is also done gradually, starting from unit tests and finishing with end-to-end system tests.

There are three different environments for integration and testing:
- Laboratory.
- Cloud platform.
- Instrumented vehicle.

The laboratory is where the software is developed and first tests are carried out. At this level, modules are working independently for each other. In this way, the errors of each module are isolated from the influence of other modules and the tracking of errors is easier. Developers commit their changes in the software modules to a version control repository in a cloud platform. This cloud platform is used both as a repository and a testbed. Each time a developer changes the software stored in the repository, it is automatically tested according to the programmed unit and integration tests. Finally, the system is tested in a real scenario using an instrumented vehicle.

The following figure summarises the planned integration and testing cycle:

Even though the unit and integration tests can be triggered automatically using the Continuous Integration tool, the whole process of integration is supervised by the Integrator (see roles defined in Section 3.3). Integrating all INLANE modules is a very complex task that cannot be relied entirely on an automated tool.

5.2 Cloud platform

5.2.1 Specifications

We are using an Amazon EC2 instance with the following characteristics:
<table>
<thead>
<tr>
<th>Instance type</th>
<th>Vcpu</th>
<th>ECU</th>
<th>Memory (GiB)</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2.medium</td>
<td>2</td>
<td>Variable</td>
<td>4</td>
<td>1TB</td>
</tr>
</tbody>
</table>

The EC2 instance was designed to be enough sized and compliant with respect to the real envisaged architecture described in Section 6.3. It was also designed to be enough to support a software version control system with Continuous Integration capabilities. The instance has an Ubuntu 16.04 image with GitLab, RTMaps and Docker installed.

### 5.3 Integration tools

#### 5.3.1 Docker

A very usual issue arises when integrating software modules developed in distributed teams. This issue is colloquially known as *dependency hell*, and reflects the frustration of system integrators that have to deal with dependencies on specific versions of software packages.

In case of INLANE, to obtain a smooth continuous integration we need to have the same software configuration in every hardware: from development PCs to the in-vehicle computer. In a traditional workflow, if someone installs a new dependency, it needs to be communicated to all developers and integrators and everybody needs to install manually the new dependency. Docker\(^4\) is a platform for building, shipping and running applications, which was designed to avoid dependency hell. It provides a way to isolate our application to the underlying host and put inside a container everything that the application needs to run. Therefore, it helps with portability and continuous integration.

Docker images are comprised of multiple layers. Each layer is just another image. A container instantiates an image and adds a top writeable layer. Parent images are read only and all changes are made at the writeable layer. Docker containers are not tied to any specific infrastructure: they run on any computer, on any infrastructure, and in any cloud.

In INLANE’s approach, a single Docker image is created, that has Ubuntu 16.04 as base image and all dependencies added in further layers. Then, each software developer or integrator can create a Docker container starting from this image. This container will wrap up everything necessary to run any software module (see Figure 8 Docker container in INLANE).

![Docker container in INLANE](image)

Following this approach, if somebody has a new dependency to install, a new image layer will be created and uploaded to the repository, so everybody could get updated easily (see Figure 9). Docker images are created and maintained under the supervision of the WP4 leader. As Docker images are read-only, developers cannot install new dependencies without informing WP4 leader.

---

\(^3\) Here ECU means Amazon’s EC2 Compute Unit. More info here: https://aws.amazon.com/ec2/faqs/?nc1=h_is

\(^4\) https://www.docker.com/
5.3.2 GitLab CI

GitLab includes a Continuous Integration tool, called GitLab CI. This tool has the following features:

- **Multi-platform**: you can execute builds on Unix, Windows, OSX, and any other platform that supports Go.
- **Multi-language**: build scripts are command line driven and work with Java, PHP, Ruby, C, and any other language.
- **Stable**: your builds run on a different machine than GitLab.
- **Parallel builds**: GitLab CI splits builds over multiple machines, for fast execution.
- **Realtime logging**: a link in the merge request takes you to the current build log that updates dynamically.
- **Versioned tests**: a `.gitlab-ci.yml` file that contains your tests, allowing everyone to contribute changes and ensuring every branch gets the tests it needs.
- **Pipeline**: you can define multiple jobs per stage and you can trigger other builds.
- **Autoscaling**: you can automatically spin up and down VM’s to make sure your builds get processed immediately and minimize costs.
- **Build artifacts**: you can upload binaries and other build artifacts to GitLab and browse and download them.
- **Test locally**: there are multiple executors and you can reproduce tests locally.
- **Docker support**: you can easily spin up other Docker containers as a service as part of the test and build docker images.

GitLab CI is the chosen tool to implement the Continuous Integration approach explained previously. GitLab CI is part of GitLab and it manages projects/builds. In order to process the builds, an additional application, GitLab Runner, is necessary, which will be deployed separately and works with GitLab CI through an API. The architecture of GitLab’s Continuous Integration approach is depicted in the following figure:
In case of INLANE, the Runner Server and the GitLab server are installed in the same server: the Amazon EC2 instance described in Section 5.2.

### 5.4 Testing strategy

The INLANE testing strategy includes unit tests, integration tests and end-to-end system tests. The overall goal of the tests is to get a quick response on the status of the software and to ensure that new developments integrate into the overall system.

#### 5.4.1 Unit tests

The objective of INLANE’s unit testing is to isolate each component of the architecture and to prove that it functions “correctly”. An RTMaps component is understood as unit. Each unit test consists of one or more RTMaps diagrams that test the corresponding components. The unit tests can be run both manually by the developer, or automatically by the continuous integration tool, which runs the tests every time somebody updates a component in the remote repository.

Each component developer contributes to the test plan by designing a testing protocol for his component. A passed test shall at least ensure that every module

- fulfills the interface requirements,
- responds correctly to all kind of inputs,
- and achieves the general result that is desired.

Each component test may contain several sub-tests.

A test is defined by:

1. **A test input generation component**: The input of each component will be either raw data recorded from a sensor or the output record of a previous module in the architecture. Some tests may use artificial data sets to simulate specific test cases.
2. **The component under test and its properties**;
3. **The unit test component**: A component that takes the outputs of the module under test as inputs. It transforms the outputs to a testing result by comparing them with the expected outputs as defined by the developer. This comparison leads to the final test result which is either passed or failed.

These three elements are arranged and connected in an RTmaps diagram. This diagram is executed and the output (log of the unit test component and RTMaps console log) is then passed to the integration tool. The integration tool will then produce a unit test report.
5.4.2 Integration tests

After having passed the unit tests, the components are put together in an incremental manner. After adding a component or a component group an integration test is performed. Starting from the components interfacing the sensors or other external data sources (beginning of the processing chain), components along the processing chain are added until the overall architecture is established. This incremental approach facilitates the tracking of errors.

The integration tests shall verify that:
- The components still work correctly after integrating them with the other system parts (instead of only working in isolation as verified by the unit tests);
- The components interact correctly: data is exchanged correctly, the components run fast enough to not interfere with downstream components.

An integration test is defined by:
1. **A test input generation component**: The input of each component group are either raw data recorded from a sensor or the output an already integrated component group. Some integration tests may use artificial data sets to simulate specific test cases.
2. **The component group to be integrated** and the properties of the modules, as well as the connections in the diagram;
3. **The integration test component**: A component that takes the outputs of the component group under test as inputs. It transforms the outputs to a testing result by comparing the outputs with the expected outputs as defined by the developer. This comparison leads to the final test outcome, which is either passed or failed.

Each integration test is represented by an RTMaps diagram. Once the unit tests have been carried out, the integration tests will be executed.
6. Performance assessment Methodology

Once the modules are integrated, the completely integrated and functionally tested system is assessed for performance. This performance assessment will be carried out according to EN 16803-1. In the end-to-end system tests, the RTMaps diagram that contains the complete INLANE solution is used.

6.1 Performance Assessment according to EN 16803

The performance assessment follows the EN standard 16803 (Space - Use of GNSS-based positioning for road Intelligent Transport Systems (ITS) - Part 1: Definitions and system engineering procedures for the establishment and assessment of performances). EN 16803 introduces the terms Positioning-based road ITS system and GNSS-based positioning terminal (GBPT). According to the standard, the entire INLANE solution is a positioning-based road ITS system. INLANE’s positioning component is a GBPT as defined by EN 16803. The output of the GPBT is position, velocity and time (PVT) information, which is the base for the road ITS application.

EN 16803 proposes performance features that can be used for performance assessment: accuracy, availability, integrity and timing. The characterization of these features shall be done by comparison of the GBPT trajectory with the trajectory of a measurement mean capable of delivering accuracy performances better of at least one magnitude compared to the GBPT. For every feature, several performance metrics are proposed for performance characterization. The results of the performance assessment shall be provided along with a description of the operational scenario and a GNSS environment characterization.
As the definition of the performance of Road ITS Applications can vary for every use case, no performance features and metrics are explicitly defined by EN 16803. These performance features are highly application-specific. A performance characterization shall be done by means of a sensitivity analysis. The idea is to replace the PVT output of the GBPT by artificially degraded trajectories (using a PVT error model, which degrades the reference trajectory) as input for the road ITS application. The application metrics are then computed using the degraded application quantity (output of the application based on degraded input) and the reference application quantity (output of the application based on reference input) leading to a characterization of system end-to-end performances.

In INLANE the standard will be applied as follows:
- Characterization of the positioning component performance in compliance with EN16803:
- Research on assessing the end-to-end INLANE system performance based on a sensitivity analysis as proposed by EN16803: in particular, this research should address the definition of use cases with Key Performance Indicators and corresponding metrics.

### 6.1.1 Performance Assessment of the Positioning Component

The basis for the PVT performance assessment is the comparison of the INLANE trajectory with a ground truth trajectory (determined with a positioning system with higher level accuracy). For every epoch, the differences between the position output by the INLANE positioning component and the reference position is computed. The statistics of those differences are the input to the performance assessment. As defined by EN 16803-1 the 50th, 75th and 95th percentiles of the cumulative distribution of errors will be computed. The transformation of the errors (e.g. local-level frame or a trajectory-related system) leads to a better interpretability.

<table>
<thead>
<tr>
<th>Output</th>
<th>Performance Feature</th>
<th>Performance Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>north, east, vertical position error</td>
<td>50th, 75th and 95th percentiles</td>
</tr>
<tr>
<td></td>
<td>along track and across track position errors</td>
<td>50th, 75th and 95th percentiles</td>
</tr>
<tr>
<td>Velocity</td>
<td>north, east, vertical velocity error</td>
<td>50th, 75th and 95th percentiles</td>
</tr>
<tr>
<td></td>
<td>along track and across track velocity errors</td>
<td>50th, 75th and 95th percentiles</td>
</tr>
<tr>
<td>Attitude</td>
<td>roll, pitch and yaw errors</td>
<td>50th, 75th and 95th percentiles</td>
</tr>
</tbody>
</table>

Table 1: Accuracy performance features and metrics for the positioning component output

The availability of the PVT information is another important performance feature of the positioning component. The availability of an output quantity is defined as the percentage of time intervals of specified length during which the positioning terminal provides at least one position output [EN16803]. The time interval has to be specified according to the requirements of the application.

In addition of this global availability feature, another feature (more specific to the applications designed) can be estimated, referred to as T-interval availability. This seems to be applicable for INLANE to the navigation application, but not really for the mapping application.

The information on accuracy and availability must be complemented by a standard-compliant description of the operational scenario and the GNSS environment.

At later stages of the project, the assessment will also include integrity and timing performances.

### 6.1.2 Performance Assessment of the Positioning-based End-to-End applications

EN 16803-1 foresees that the performance assessment of the positioning-based end-to-end applications (here: lane-level navigation and crowd-based map generation) is performed by means of a sensitivity analysis. The following steps are prescribed:
1. Definition of the operational scenario
2. GBPT field test execution
The core of this procedure is the PVT error model identification and the generation of the degraded trajectories. Methods for doing this are topic to current research. For GNSS-only positioning systems error models have been developed by several authors\(^5\). Modelling the errors of a positioning system that relies heavily on sensor data fusion (GNSS, visual and inertial data) has not been addressed yet, so this is an opportunity for the later stages of the INLANE project.

### 6.2 Test scenarios

#### 6.2.1 Datasets for laboratory tests

Several datasets have already been recorded using RTMaps. The trajectories are visualized below.

![Figure 14 Recorded datasets (yellow, orange, blue)](image)

Total length of the datasets is 111 km and total size of 550 GB (mostly video data).

As the in-vehicle set-up is not finalized, the current datasets includes the following:

- Stereo camera (Pointgrey Bumblebee2) 1024x768, 20 Hertz, HFOV 40 degrees.
- UBlx GNSS receiver 1 Hertz.
- XSens VRU 20 Hertz (internally 200 Hertz).
- Mobileye C2-270 (object and lane data).
- Vehicle ECU data (vehicle ego-state data).

#### 6.2.2 Scenario of field tests

At the DITCM test site the evaluation metrics related to the accuracy of vehicle positioning will be determined. This will involve driving the vehicle in different traffic scenarios and circumstances, including:

\(^5\) See http://www.sappart.net/?page_id=423
• High-way scenarios
• Entry and exit ramp scenarios
• Traffic light stop-and-go scenarios
• Intersection crossing scenarios, including left, right, and full turns
• Dense traffic urban scenarios
• Tunnels with underground exits scenarios (not in DITCM)
• Low-visibility scenarios (potentially night-time, rain, fog, and snow).

The distance driven per scenario will differ but the aim is to drive 100 Km in total over the duration of the INLANE project.

6.3 Hardware for field tests

The figure below shows the in-vehicle hardware that will be available. This set-up is still under development but the following sensors are already functional:

- Stereo camera (Pointgrey Bumblebee2) 1024x768, 20 Hertz, HFOV 40 degrees.
- UBlx GNNS receiver 1 Hertz.
- Xsens VRU 20 Hertz (internally 200 Hertz).
- Mobileye C2-270 (object and lane data).
- Vehicle ECU data (vehicle ego-state data).

![In-vehicle hardware diagram](image)

**Figure 15 In-vehicle hardware.**
7. Year 1 testing results

The performance assessment concept described in this document was part of the development in year one. The first versions of the system components have not been evaluated in this unified manner yet. To give an overview of the current performance of every module, the current test results are resumed. Further details can be found in the corresponding deliverables, where in-depth discussions are included.

7.1 Sensor Data Fusion

The developed sensor data fusion component has been tested in different operational scenarios. The current results are based on low-cost IMU and GPS sensors. Tests in open sky scenarios demonstrate a good performance. The system already meets the requirements for lane-level navigation here (accuracy of 0.5 m RMSE for North and East respectively). The accuracy in urban scenarios is not at this level yet, although the basic sensor fusion approach is able to improve the positioning performance in terms of accuracy and availability significantly. The main problems arise if the GNSS is unavailable for a long-time span. Because of the IMU sensor errors, long GNSS gaps cannot be bridged properly. The accuracy degrades quickly in these cases.

The complete testing results can be found in deliverable D2.1.

7.2 Sensor-to-Map Alignment

In the course of first evaluations, the KITTI dataset was used, which consists of vision, IMU and GNSS data. The map data used in the evaluation process is obtained from the OpenStreetMap project and manually enhanced by additional details such as lane connections and detailed geometry. It can be seen that roads with detectable markings result in a successful match. Alignments at intersections not only lead to a lateral but also to a longitudinal alignment. Even when dealing with environmental influences such as shadows, true matches can be achieved. Despite the robustness demonstrated in several scenarios, heavy deviations in the map from the true road shape can impede an alignment. Sources of errors are parking or moving cars limiting the view, neighbouring lanes as well as other dominant edges, such as train rails or fences. Missing lane markings and incorrectly estimated lane width remain a problem to be tackled as well. At his moment, the implementation is not yet real-time capable.

The complete testing results can be found in deliverable D2.5.

7.3 Visual Odometry

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.

The complete testing results can be found in deliverable D2.3.

7.1 Lane Detection

The vision-based Lane Detection algorithm is a very important component of INLANE’s architecture and consequently, it has been thoroughly tested during the first year of the project. To evaluate its performance, a number of metrics have been defined:
• Detection performance in terms of False Positive, False Negative, True Positive: overall, the results are close to the requirements in almost all cases.

• Lane width: the mean error of the lane width clearly satisfies the requirements in all videos and curvature ranges.

• Lane curvature: lane curvature errors are almost negligible in all cases, satisfying required values with a large margin.

• Distance to lane boundaries: distances to both left and right edge of the lane are precisely estimated, with mean errors of less than 7 cm.

• 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.

• Computational cost: the computational cost of the algorithm is very low, and it can run real-time in most standard PCs.

The complete testing results and requirements definition can be found in deliverable D2.3.

7.2 Traffic Sign Recognition

For the classification stage, evaluation metrics have been obtained using the popular German Traffic Signs Recognition Benchmark Database (GTSRB). Considering the considered categories, the trained model achieves ~ 90% of accuracy over the GTSRB test set. 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. The combined general system performance has also been assessed subjectively, and satisfactory results have been observed. Regarding processing times, the obtained results are reasonable for a first prototype, however further optimizations are necessary.

The complete testing results can be found in deliverable D2.3.

7.3 Static map

Maps used in the INLANE project are based on NDS. To make these maps suited for lane level navigation functionality, the NDS specification map is extended with extra attributes for precise localization and lane level navigation. The extra attributes are organized in NDS as extra layers stacked onto the maps currently used in production maps (SD maps). New tooling is created to help the creation of the proprietary run-time map files in a semi-automated process. The HD map prototypes are created for the test track in Eindhoven. In a next step the access library will be extended with 3D attributes (RoadDNA) for accurate localization and will be made available for the main stream prototype development with the localization partners. It is still under discussion whether the access library will be implemented as an RTMaps module or if RTMaps can use the library via UDP/TCP or similar.

Further information is detailed in D3.1.

7.4 Local Dynamic Map

The Local Dynamic Map (LDM) is a key element for the sensor-to-map alignment process. The LDM
contains and maintains a consistent online world model comprising all behaviourally relevant elements in the immediate environment of the car. For INLANE, the LDM is used to store and retrieve the road structures on a lane geometry basis. These are then used to generate a virtual view of the surrounding, which can afterwards be compared to the camera image for an accurate positioning relative to the road structure.

A prototype for an LDM was programmed on basis of a noSQL/graph database server, since the natural structure of the road elements lends itself to a graph-based representation. A binary/python interface and low level API regulates the client-to-server communication, so that the LDM information can be accessed from any INLANE module. A prototypical representation scheme has been created for the necessary road geometry elements, and a toolchain has been set up which allows the import and the lane-accurate map enrichment from standard map data. Currently, we are using OpenStreetMap data for this process, but an interface to the NDS and especially the NDS Open Lane Model 1.0 map representation formats is under discussion (see section 7.3). Along the same line, we are currently checking consistency with the concepts for LDMs as put forward in the ISO and ETSI standardization committees.

First tests with the implementation have confirmed the suitability of the tool chain and the graph-based LDM approach for the sensor-to-map alignment module. Next steps will be the specification of a representation schema for non-road traffic elements as well as tests on latency and access performance.

7.5 Navigation application

A first prototype of a lane navigation application was built. It consists of an engine part implementing the generic navigation functions and an application part implementing the HMI.

An automated build-environment is created for the lane navigation engine with support for the mainstream intel-based prototyping platform and for the embedded tablet platform. For these platforms, the SDK are made available for INLANE experiments.

In a next step the environment will be extended with support for the embedded platform suited for automated driving experiments.

Further information is detailed in D3.1.
8. Conclusions

A common methodology and set of rules to efficiently develop, integrate and evaluate the different components involved and the resulting system is mandatory when distributed teams with heterogeneous backgrounds work together. This document tries to define all the necessary guidelines required in the INLANE project. We have based our integration methodology on the state-of-the-art practice called Continuous Integration. We believe that this approach improves the traditional integration workflow mitigating the most common integration problems. We have also set some good practices for software versioning, version control and documentation, and we have defined some common software development tools.

INLANE’s testing strategy includes unit tests, integration tests and end-to-end system tests. Although this test classification may seem obvious, tests are not always conducted correctly and in a balanced way in innovation projects. The tendency is to go from unit tests to system tests directly, sometimes even skipping the necessary unit tests. This increases dramatically the number of costly end-to-end system tests and the total expense of the prototype integration. In INLANE we are aligned with the Continuous Integration philosophy and we promote unit tests as the bulk of our tests. Then we plan a lower number of integration tests, and finally, a few end-to-end system tests including field tests.

INLANE proposes a GNSS-based positioning system for vehicles, so we have designed INLANE’s performance assessment methodology following the corresponding European standard (EN 16803). The first versions of the positioning system have not been evaluated in this unified manner yet, but we plan to do it in the next cycle. In this document, we summarize the performance results of each software component obtained during the first year of the project using different assessment metrics. In addition of the positioning system itself, the standard suggests also end-to-end application tests, currently planned in the latest cycle of the project. Here, the application tests start from the same inputs (sensors) as the positioning system, but extend up to navigation guidance and map update generation.

To sum up, we have defined INLANE’s development, integration, testing and validation methodology based on other methodologies and good practices that have already been applied successfully in many other projects, and we have followed the corresponding standards when applicable. The next iteration of the document (D4.2) will include the end-to-end and positioning system performance results obtained from the planned field tests.