# System Specification - Cooperative Research Platform

Gergo Ferenc Igneczi[1*], Dávid Józsa[1†] and Mátyás Mesics[1†]

[1]Vehicle Industry Research Center, University of Győr, Egyetem sq., Győr, 9024, Győr-Moson-Sopron, Hungary.
[2]ZalaZONE Innovation Park, University of Győr, Dr. Michelberger Pál str., Zalaegerszeg, 8900, Zala, Hungary.

*Corresponding author(s). E-mail(s): gergo.igneczi@ga.sze.hu;
Contributing authors: jozsa.david@ga.sze.hu; mesics.matyas@ga.sze.hu;
†These authors contributed equally to this work.

**Abstract**

System developed, owned and maintained by University of Győr to accomplish various automated driving function tasks.

**Keywords:** none

## 1 Goal of the document

This document summarizes the details of how the Cooperative Research Platform (CRP) is built up. It consists of the multiple layers:

- System Functionality,
- Architecture,
- Scenario Coverage and demonstration basis

# 2 Supplements

## 2.1 Corresponding terminology

- E2E: end-to-end, usually referred to as standalone operation of a function, without the need of e.g., pre-recorded data, and this function can be used by a non-technician user
- function: practical manifestation of technical implementation
- architecture: collection of components that are arranged into a pre-defined structure,
- ground architecture: white-paper definition of system components, without dependencies like Autoware,
- function architecture: real structure of the system components, that are directly usable in the vehicle.

## 2.2 Coordinate Frames

## 2.3 Nomenclature

- $\alpha_f$ - front road-wheel angle
- $\delta$ - position offset to the centerline of the lane
- $\psi$ - yawrate of the vehicle (rotational speed around $\zeta$ axis)
- $v_\xi$ - longitudinal speed of the vehicle, in the vehicle frame $(\xi, \eta, \zeta)$
- $a_\xi$ - longitudinal acceleration of the vehicle, in the vehicle frame $(\xi, \eta, \zeta)$
- $L_w$ - wheel base
- $\theta$ - orientation of the vehicle in the global $(x, y, z)$ coordinate frame

## 2.4 Testing Concept

Implement function code in function layer, then integrate to application layer. At this stage, record raw data (mcaps) together with vehicle and controller integration layers. The resulting measurement file can be used for open-loop tests, that is satisfactory except for vehicle control components.

# 3 Function Specification

This Section describes the high-level specifications of the covered functionality. Autoware architecture:
https://app.diagrams.net/?lightbox=1#Uhttps%3A%2F%
2Fautowarefoundation.github.io%2Fautoware-documentation%
2Fmain%2Fdesign%2Fautoware-architecture%2Fnode-diagram%
2Foverall-node-diagram-autoware-universe.drawio.svg

## 3.1 Intelligent Speed Adjustment

- Step 1 functionality: longitudinal speed control adjusted to static information, such as curve and local regulations (speed limit).
- Step 2 functionality: step 1 + speed adjustment on dynamic information, such as moving objects (e.g., followed vehicle).

For both: speed range is $0.0 \leq v_x \leq 150.0kph$, which therefore includes automatic start/stop functionality. Function is illustrated in Figure 1.

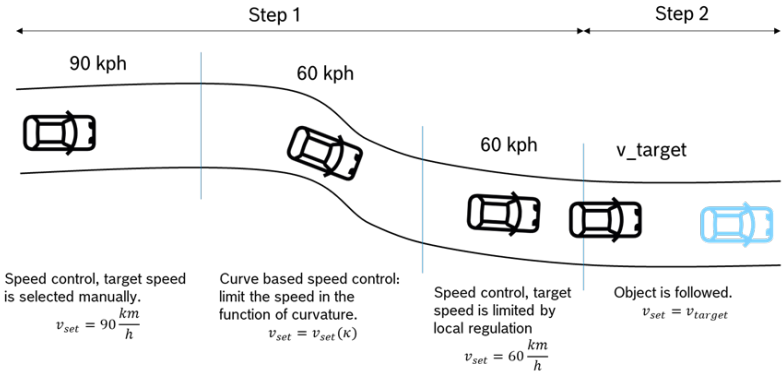


**Fig. 1**: Function illustration, both step 1 and step 2 functionality.

## 3.2 Longitudinal Emergency Function

Functionality: vehicle or delegated sensors provide information about static / dynamic objects. The function decides proper strategy to stop the vehicle (and where to stop it). Then, this strategy is accomplished by applying proper braking force. Function use cases are shown in Figure 2. Operation range: $0.0 \leq v_x \leq 150.0kph$.

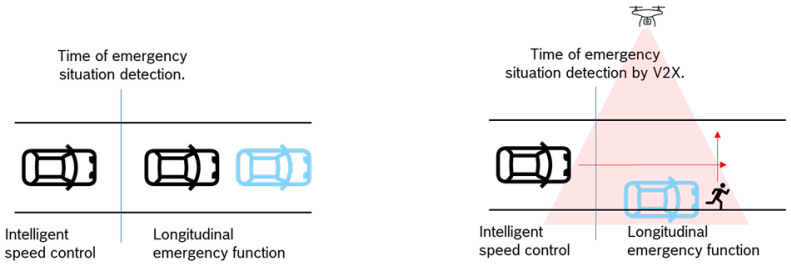


**Fig. 2**: Longitudinal emergency use cases from vehicle sensors and infrastructure sensors.

## 3.3 Lane Follow

- Step 1 functionality: vehicle is running in a lane, which is bounded by lane edges (markers or only the edge of the drivable surface) and the vehicle follows the centerline of the lane (or externally defined local trajectory). Operation range: $a_{y,max} = 5m/s^2$ , $0.0kph \leq v_x \leq 150.0kph$, $\sigma^2_{e_y} \leq 0.1m$.

- Step 2 functionality: Drivable corridor is shifted due to e.g., temporarily shifted road works, which is bounded by 3D obstacles like cones, walls...etc. Vehicle (with lower dynamics) can still navigate through this drivable corridors. Operation range: $a_{y,max} = 3m/s^2$ , $0.0kph \leq v_x \leq 110.0kph$, $\sigma^2_{e_y)} \leq 0.1m$.

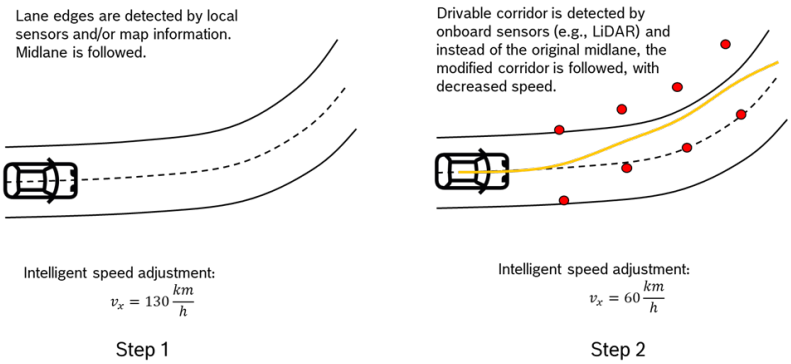


**Fig. 3**: Lane follow functionality steps and covered operation.

# 4 Architecture Specification

## 4.1 Comprehensive notes

The architecture is defined based on the E2E function specifications. There are two main concepts that must be considered at all time:

- fulfill E2E function requirements with the least architecture components,
- re-use Autoware components where possible, but keeping its number (or number/-function) as low as possible.

Therefore, the work model shown in Figure 4 is strongly recommended. This concept may hinder the efficient/reliable planning of tasks regarding architecture definition, but ensures that the above two concepts are respected.

## 4.2 General Architecture

### 4.2.1 High-level Black-box architecture

### 4.2.2 Gray-box architecture

### 4.2.3 Functional (ROS2) architecture

## 4.3 Function Architecture Specification

### 4.3.1 Intelligent Speed Adjustment

Step 1 architecture is shown in Figure 8. Note: even if we only control the vehicle

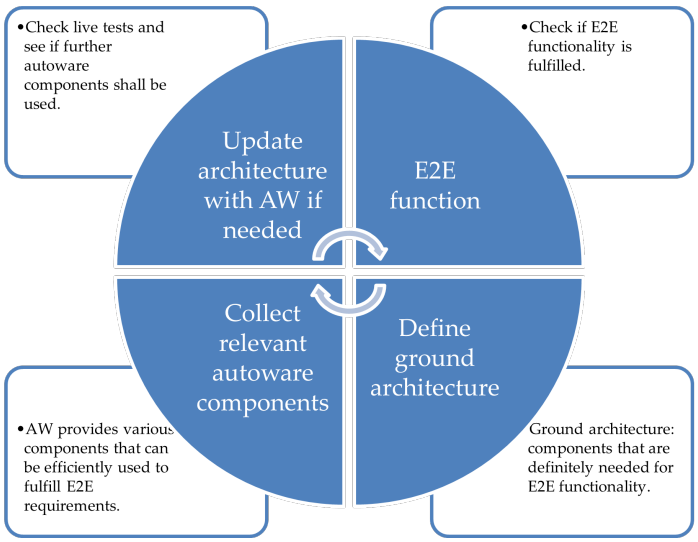


**Fig. 4**: Working cycle - proposal

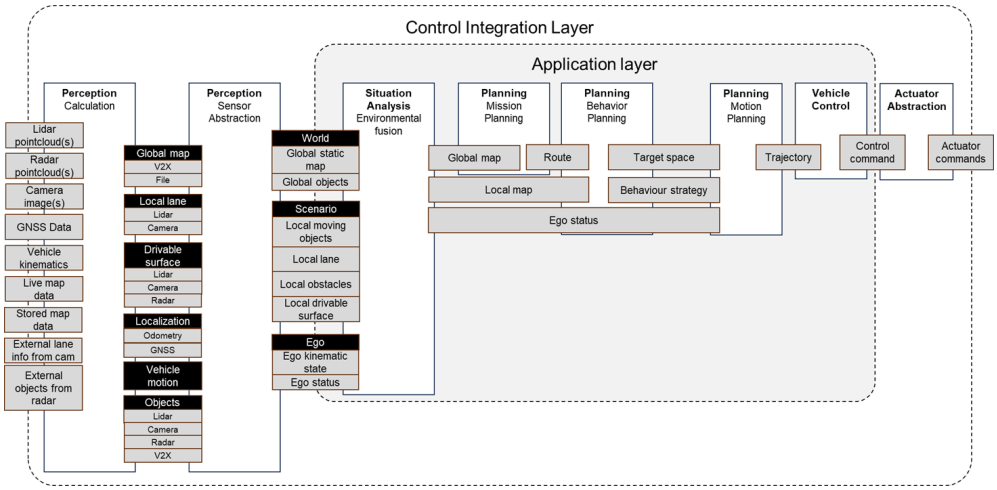


**Fig. 5**: Black box architecture

longitudinally, the lateral path shall be filled with dummy values. Idea: add a straight line with no offset. Later, it must be solved that the vehicle is longitudinally controlled by the system, but laterally by the driver.
Step 2 architecture is shown in Figure 9.

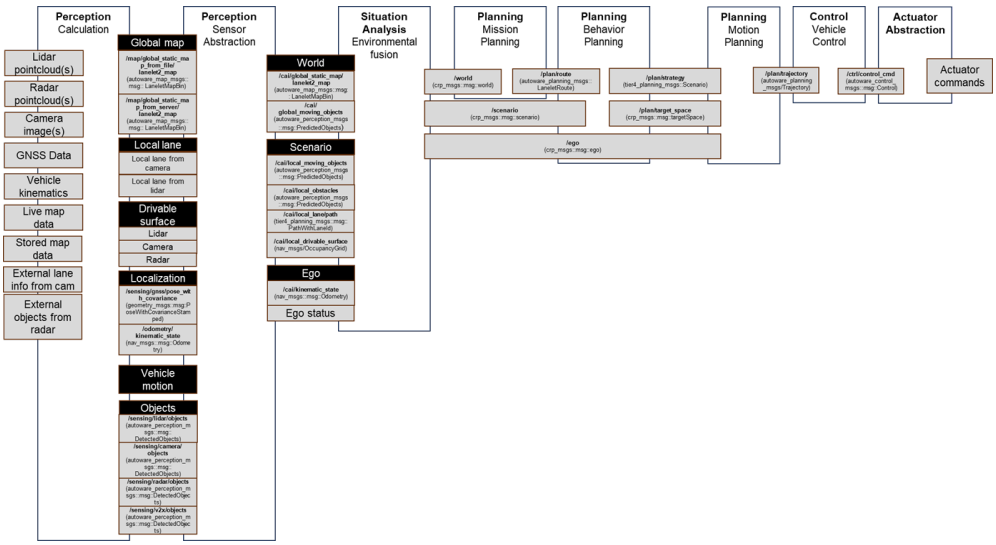


**Fig. 6**: Gray box architecture.

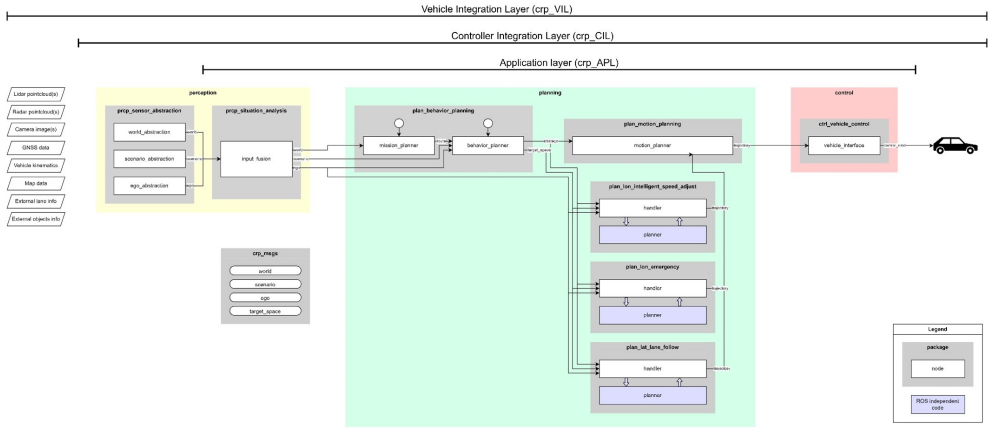


**Fig. 7**: Functional architecture

### 4.3.2 Longitudinal emergency function

Based on distributed sensor data calculate the trigger of the emergency scenario. Corresponding architecture is shown in Figure 10.

### 4.3.3 Lane Follow

Lane follow functional architecture is shown in Figure 11. Lane follow architecture consists of new components of path planner, that takes geometry information of the road, and plans a smooth (drivable) path, which is then handed over to the lateral

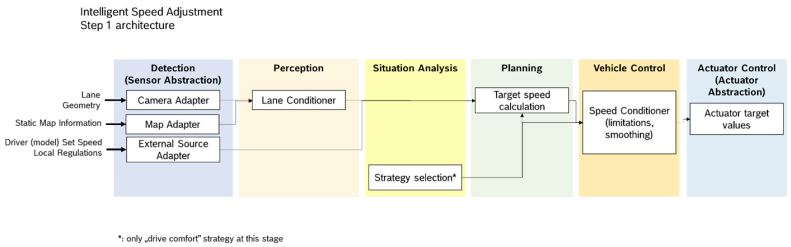


**Fig. 8**: Architecture components of step 1 functionality of ISA

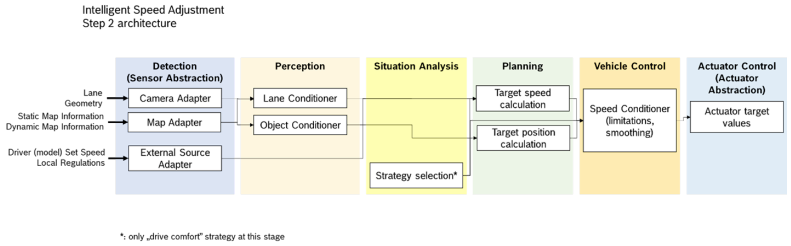


**Fig. 9**: Architecture components of step 2 functionality of ISA

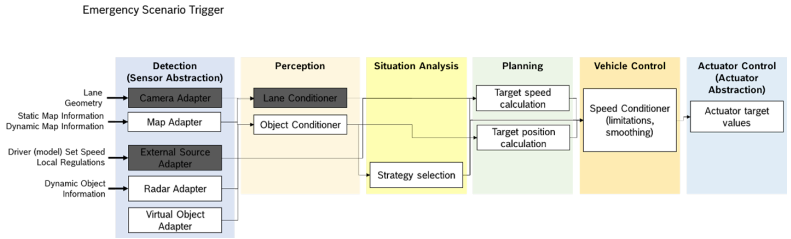


**Fig. 10**: Architecture components of step 1 longitudinal emergency function.

controller component. This controller controls only the lateral movement of the vehicle, producing output to the actuator control (i.e., steering angle). Note: route input comes from mission planner, which is currently not part of the architecture. During integration process, it may be extended. Step 2 architecture is shown in Figure 12.

# 5 Message Definitions

## 5.1 CRP messages

### 5.1.1 crp_msgs/msg/scenario

This interface holds information of four main types:

- local moving objects: highest layer which is associated with other objects that move around the ego vehicle, such as other vehicles, pedestrians, animals...etc.

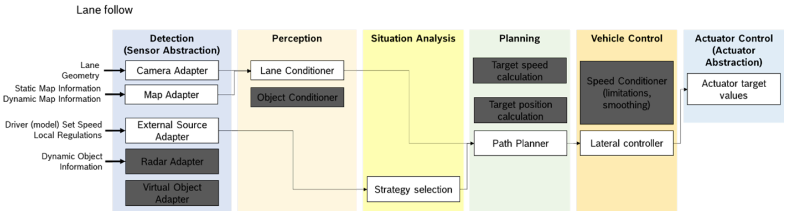


**Fig. 11**: Architecture components of step 1 functionality of LF (lane follower) without behaviour layer.

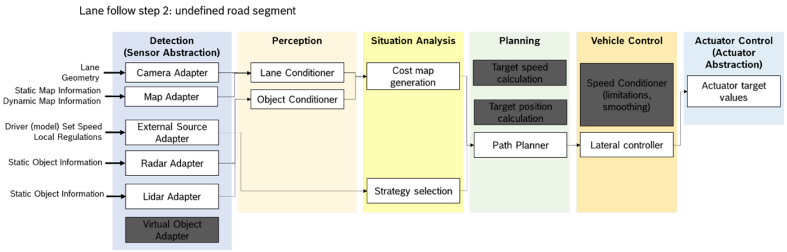


**Fig. 12**: Architecture components of step 2 functionality of LF (lane follower) without behaviour layer.

- local obstacles: static items that are located around the ego vehicle.
- local lanes: the lanes that are mainly marked by painted markers and form the static driving corridors.
- local drivable surface: the most indefinite representation of the local environment, in the form of a generic occupancy grid.

This interface type (in contrast to globally defined 'world' interface) must contain data with as high accuracy as possible. Message definition:

```
std_msgs/Header header

autoware_perception_msgs/PredictedObject[] local_moving_objects
autoware_perception_msgs/PredictedObject[] local_obstacles
tier4_planning_msgs/PathWithLaneId[] lanes
// traffic rules information to be added
nav_msgs/OccupancyGrid free_space
std_msgs/Float32 maximum_speed
```

Note: the traffic rules collect all types of information that are coming from the static rules and can impact the selected bahviour. These are like:

- stop lines (stop signs and lines)
- speed limitation
- traffic light information (semi-static)...etc.

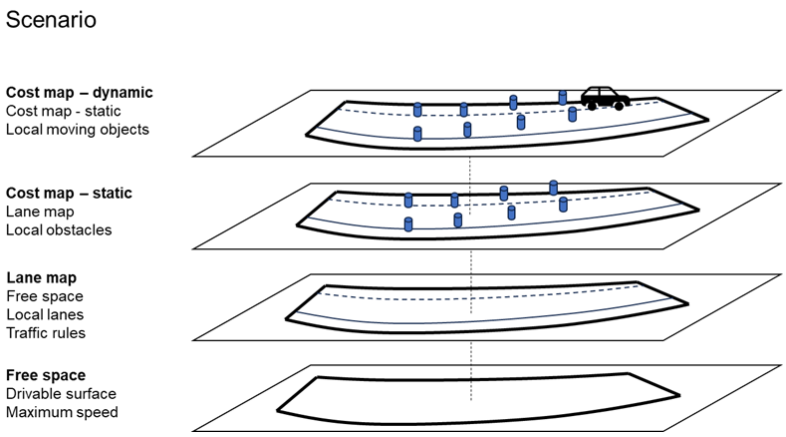


**Fig. 13**: Illustration of the scenario layers.

### 5.1.2 crp_msgs/msg/world

### 5.1.3 crp_msgs/msg/ego

This interface contains every relevant information about the ego vehicle:

- pose: position, orientations and uncertainty
- velocity: linear and angular
- acceleration: linear and angular
- wheel angle

Message definition:

```
std_msgs/Header header

geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
geometry_msgs/AccelWithCovariance accel
float32 wheel_angle
```

## 5.2 Tier4 messages in use

### 5.2.1 Path

This is a tier4 autoware message extension, with the following definition:

```
std_msgs/Header header

tier4_planning_msgs/PathPoint[] points
nav_msgs/OccupancyGrid drivable_area
```

### 5.2.2 Path point

```
uint8 REFERENCE=0
uint8 FIXED=1
geometry_msgs/Pose pose
geometry_msgs/Twist twist
uint8 type
```

### 5.2.3 Trajectory

```
std_msgs/Header header
tier4_planning_msgs/TrajectoryPoint[] points
```

### 5.2.4 Trajectory point

```
geometry_msgs/Pose pose
geometry_msgs/Twist twist
geometry_msgs/Accel accel
```

# 6 Using the Platform

## 6.1 Installation (source)

Create workspace and clone the repository:

```
mkdir -p crp_ws/src
cd crp_ws/src
git clone https://github.com/jkk-research/CooperativeResearchPlatform.git
    --recurse-submodules
```

Install dependencies and build the workspace:

```
cd ..
rosdep install --from-paths src --ignore-src -y
colcon build --symlink-install
```

## 6.2 Using it in a docker container

### 6.2.1 Docker container

The platform can be used in a docker container. The docker is available here: https://hub.docker.com/repository/docker/anonymdavid/crp/general

### 6.2.2 Building the docker

Alternatively the autoware docker can be built using the script inside the autoware repository. The docker container is built on top of the autoware docker (ghcr.io/autowarefoundation/autoware:latest-devel-cuda). Autoware and CRP should be built for the same platform type (amd64/arm64).
The container can be build to ARM64 platform:

```
cd CooperativeResearchPlatform
./docker/build_arm64.sh
```

### 6.2.3 Running the docker

The docker can be run with the following command:

```
docker run -it --rm --network host --ipc host --pid host --device
    /dev/leaf0 --device /dev/leaf1 crp_arm64:latest
```

This way the docker shares the network with the host and has access to the kvaser CAN device.

# 7 Coding rules

## 7.1 Naming convention

- Use camel case everywhere if not stated otherwise - example: *latLaneFollow*
- Classes and structs should start with upper case letter - example: *MotionHandler*
- Methods should be camel case (starting with small case) - example: *scenarioCallback*
- Method arguments have no prefix/suffix, they must be given with camel case - example: *plannerInput*
- Constant variables should be all capital letters. Instead of camel case the words should be separated with '_' - example: *MAX_SPEED*
- In classes all member variables should start with the 'm_' prefix - example: *m_vxEgo*
- The runtime calibratable parameters should start with the 'p_' prefix - example: *p_mainThreshold*
- In special cases (e.g., subscribers/publishers) extra name tags can be added with lower case, separated by underscore - example: *m_sub_strategy_*
- Pointers should comply with the previous rules but should have a '_' suffix - example: *m_sub_strategy_*
- Namespaces should be fully lower case - example: *crp*
- Maximum namespace depth should be 3
- File and folder names should be camel case where it is not restricted (e.g. ROS2 naming rules)

## 7.2 Cplusplus

### 7.2.1 General

- Header and source files should be separated ('include' folder for headers and 'src' folder for source files)
- Header files should only contain declarations, the definitions should be in a source file that includes the header
- If applicable, include further header files in the main header file, not in the source file (source file (i.e., cpp) should only include its own declaration header)
- Functional code should be included in a separate cmake file (with .cmake extension), which is then included in the main CMakeList.txt (this way, cmake file with functional sources is ROS agnostic)

### 7.2.2 Header files

Every header file should...

- have the '.hpp' extension,
- have header guards (#ifndef, #define, #endif) and it should be all capitals,
- contain max. one class.

### 7.2.3 Source files

Every source file should...

- have the ".cpp" extension.

## 7.3 ROS2

### 7.3.1 Packages

Non-driver package names should start with and abbreviation of the following categories:

- prcp (perception)
- plan (planning)
- ctrl (control)

All dependencies must be set in the package.xml and in the CMakeList.txt.

# 8 Package documentations

## 8.1 pacmod_extender

### 8.1.1 Purpose

The purpose of this package is to extend the default pacmod capabilities on the Lexus vehicle by decoding CAN messages or calculating new data from the inputs. The package is designed to work seamlessly with the already existing pacmod3 system.

### 8.1.2 Usage

The package can be used by running the executable node. This way it uses the default namespace for subscriptions and publishers:

```
ros2 run pacmod_extender pacmod_extender_node
```

The other way is to use the launcher. This launcher is tailored for the Lexus vehicle by giving the executable the necessary namespace to match the other components:

```
ros2 launch pacmod_extender pacmod_extender.launch.py
```

### 8.1.3 IO

Input topics are given in Table 1, outputs are given in Table 2.

**Table 1**: Pacmod extender inputs.

| Data | Message name | Message Type |
|---|---|---|
| Raw can data | pacmod/can_tx | can_msgs/msg /Frame |
| Vehicle test | vehicle_status | geometry_msgs/msg/TwistStamped |

**Table 2**: Pacmod extender output.

| Data | Message name | Message Type |
|---|---|---|
| Linear acceleration | pacmod/linear_accel_rpt | pacmod3_msgs/msg/LinearAccelRpt |
| Calculated yaw rate | pacmod/yaw_rate_calc_rpt | Pacmod3_msgs/msg/YawRateRpt |

### 8.1.4 Inner workings

The main functionality is the decoding of previously not used CAN messages. The decodings are defined in the PacmodDefinitions class. Every message has a decode method that requires the CAN message as parameter. The message IDs are stored as

constants in the class. The PacmodExtender class is the main class that is executed as a node. It subscribes to the inputs, uses the PacmodDefinitions class to decode the CAN messages and outputs the new messages. The output rate of every message depends on the input frequencies. Every output value is in SI units. Decodings:

- Linear acceleration
- longitudinal, lateral, vertical acceleration in m/s

Calculations:

- Yaw rate $\psi = v_\xi tan(\frac{\alpha_f}{L_w})$, where $\psi$ is the yaw rate, $\alpha_f$ is the front road-wheel angle and $L_w$ is the wheelbase.

## 8.2 duro_gps_launcher

### 8.2.1 Purpose

The purpose of this package is to launch the duro GPS driver. The driver is in a separate repository included as a subrepository. It also contains a node (duro_topic_converter) that converts the duro GPS topics to the predefined CRP topics.

### 8.2.2 Usage

The package only contains the launcher for the driver. This can be started as follows:

```
ros2 launch duro_gps_wrapper duro.launch.py
```

### 8.2.3 Launch parameters

The parameters of the launcher are given in Table 3. The default values are set for the Lexus vehicle.

**Table 3**: Duro gps launcher parameters.

| Name | Default value | Description |
|---|---|---|
| duro_namespace | gps/duro | Namespace for the Novatel GPS |
| duro_ip | 192.168.10.11 | IP address of the Duro GPS |
| duro_port | 5555 | Port of the Novatel GPS |

## 8.3 novatel_gps_launcher

### 8.3.1 Purpose

The purpose of this package is to launch the novatel GPS driver. The driver is in a separate repository included as a subrepository. It also contains a node (novatel_topic_converter) that converts the novatel GPS topics to the predefined CRP topics.

### 8.3.2 Usage

The package only contains the launcher for the driver. This can be started as follows:

```
ros2 launch novatel_gps_wrapper novatel.launch.py
```

### 8.3.3 Launch parameters

The parameters of the launcher are given in Table 4. The default values are set for the Nissan leaf vehicle.

**Table 4**: Novatel gps launcher parameters.

| Name | Default value | Description |
|---|---|---|
| novatel_namespace | gps/nova | Namespace for the Novatel GPS |
| novatel_ip | 192.168.1.11 | IP address of the Novatel GPS |
| novatel_port | 3002 | Port of the Novatel GPS |
| novatel_imu_frame | /nissan9/nova/imu | IMU frame id of the Novatel GPS |
| novatel_frame_id | /nissan9/nova/gps | Frame id of the Novatel GPS |

## 8.4 lanelet_handler

### 8.4.1 Purpose

The purpose of this package is to load and publish a lanelet2 map to a specified topic using the autoware lanelet2 library.

### 8.4.2 Usage

The package can be used by the provided launcher:

```
ros2 launch lanelet_handler laneletFileLoader.launch.py
```

### 8.4.3 Launch parameters

Launch parameters are given in Table 5.

## 8.5 mcap_rec

### 8.5.1 Purpose

This package contains scipts for recording all the necessary topics of the project.

### 8.5.2 Usage

The package can be used by the provided launchers:

**Table 5**: Launch parameters.

| Name | Default value | Description |
|---|---|---|
| map_file_path | | Path to the lanelet2 map file |
| map_output_topic | /map/global_static_map_from_file/ lanelet2_map | Output topic for the map binary |
| map_frame_id | map | Frame id of the lanelet2 map |
| map_visualization_topic | /map/global_static_map_from_file/ lanelet2_map_visualization | Frame id of the lanelet2 map |

```
ros2 launch mcap_rec recordLexus.launch.py tag:=name_of_the_recording
```

### 8.5.3 Recorded params

Each launcher has a different preset for which topics to record. The recorded topics are defined in the etc folder in txt files.

## 8.6 nissan_can_driver

### 8.6.1 Purpose

The purpose of this package is to publish the vehicle status (speed, steering) to topics and receive control commands from topics to control the vehicle.

### 8.6.2 Usage

The package can be used by the provided launcher:

```
ros2 launch nissan_can_driver can_driver_kvaser.launch.py
```

In the current implementation a kvaser device with 2 channels is used for the communication. On the first channel the CAN data is received from the vehicle and on the second channel the control commands are sent to the controllers.

### 8.6.3 Launch parameters

Launch parameters are given in Table 6.

**Table 6**: Launch parameters.

| Name | Default value | Description |
|---|---|---|
| autoware_control_input | True | Whether to use autoware or standard type input message for control |
| kvaser_hardware_id | 11162 | HW ID of the kvaser device used for CAN communication |

### 8.6.4 IO

Input topics are given in Table 7.

**Table 7**: Launch parameters.

| Name | Message type | Description |
|---|---|---|
| /ctrl/control_cmd | autoware_control_msgs/msg/Control | Control input topic if *autoware_control_input* is set to true<br>Contains speed and steering commands |
| /ctrl/speed_cmd | std_msgs/msg/Float32 | Control input topic if *autoware_control_input* is set to true<br>Contains speed command (m/s) |
| /ctrl/steering_cmd | std_msgs/msg/Float32 | Control input topic if *autoware_control_input* is set to true<br>Contains steering command (tire angle, rad) |

# 9 Map Creation

## 9.1 Data Process

To create lane maps, the following devices were used in a single vehicle:

- Bosch MPC2.0 video camera - detecting lane edge markings [1],
- ADMA Gen3 GNSS system with RTK service - for accurate localization [2].

According to the datasheet of the devices, the camera estimates lane marking center-point position compared to the vehicle rear axis with an accuracy of $2cm$, assuming confidence of at least 75%. The confidence value is also provided by the camera and measured during our test drives. In the data process phase measurement points that have lower confidence than this threshold are neglected. Then, the points of the lane edges and centerlines are calculated. The camera returns various parameters of the lane edges. These are the following:

- $d$ - lateral position of the lane edge(s) in the ego vehicle frame (note: the ego vehicle frame is aligned with the middle point of the rear axle),
- $\Theta$ - orientation offset to the lane edge(s),
- $\kappa = \frac{1}{R}$ - curvature of the lane edge(s).

All quantities are calculated at $x = 0$ local coordinate. An illustration of the parameters is shown in Figure 14. Knowing the pose of the ego vehicle, i.e., $\boldsymbol{P}_{ego} = \left[\boldsymbol{\rho}(X, Y)\ \Psi\right]$ the global (UTM) position and orientation of the vehicle, the all-time lane edge point at $x = 0$ local coordinate can be calculated from $\boldsymbol{P}_{ego}$ and $d(x = 0)$, with the following simple 2D transformation equations:

$$X_{lane}(t) = X(t) - sin(\Psi(t))d(t) \tag{1}$$

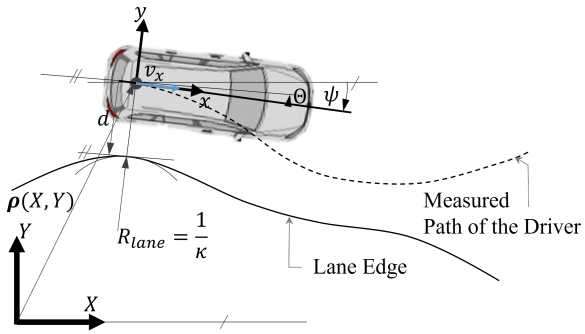


**Fig. 14**: The lane geometry relations in the ego vehicle frame.

$$Y_{lane}(t) = Y(t) + cos(\Psi(t))d(t) \tag{2}$$

The camera used in the current study can detect the lane edges of the ego lane, therefore there are $d_{left}(t)$ and $d_{right}(t)$. The lateral distance to the centerline of the lane is calculated as the average of the left and right lane edge distance values, namely: $d_{center}(t) = 0.5 * (d_{left}(t) + d_{right}(t))$.

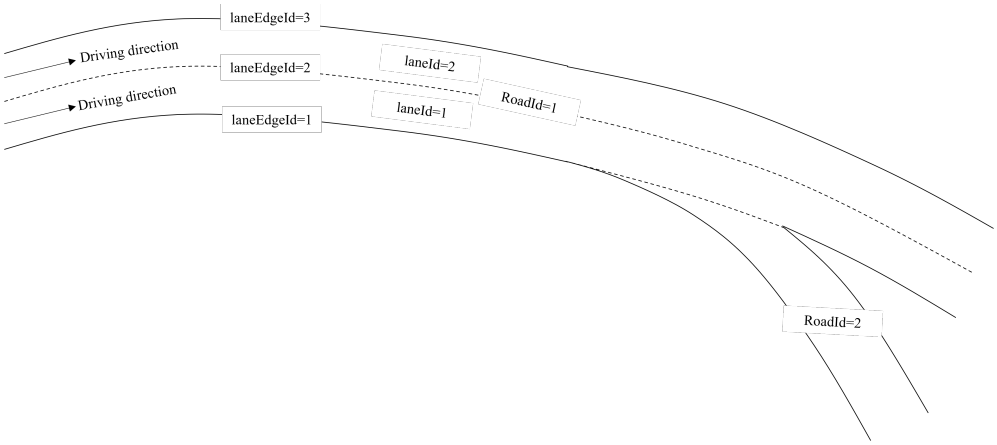


**Fig. 15**: Interpretation of the lane identification numbers - illustration only.

After the lane marking points are processed, they are saved to a compact structure. In this structure, each lane edge marking, as well as the centerline are stored separately.

| Note: the steps given in this subsection are implemented in *dataPreprocess.m* |
|---|

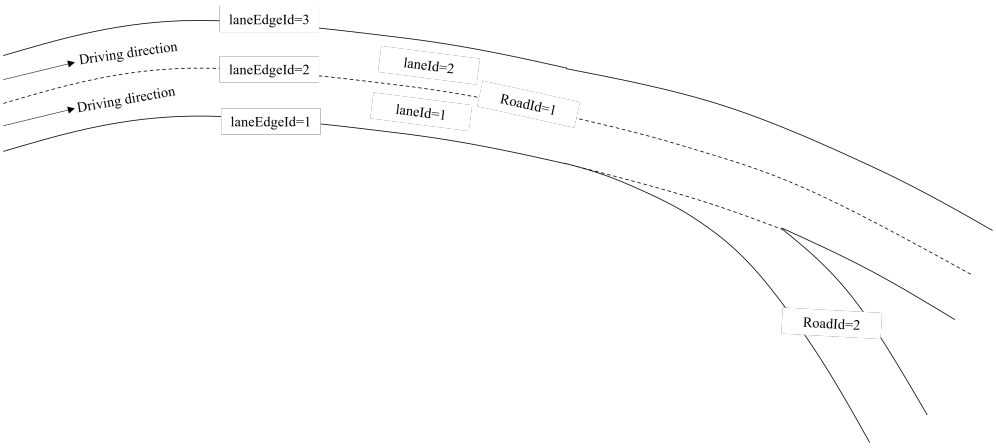


**Fig. 16**: Interpretation of the lane identification numbers - illustration only.

## 9.2 Algorithm steps

By this point, a well conditioned structure of the lane edges and centerlines are produced. The preprocess steps are summarized as follows:

1. Read raw measurement files and do the name standardization,
2. pre-filtering unreliable measurement points either due to camera maldetection or DGPS inaccuracies,
3. apply 2D transformation to calculate line points of lane edges and the centerlines,
4. match the road, lanes and lane edges to predefined identification numbers.

Out of this list, the first three steps can be done without any issues. However, the last step is difficult with no prior knowledge. E.g., how shall we know which road the vehicle is on? By knowing the road ID, how can we decide which lane the vehicle travels in (especially valid for multi-lane roads, such as highways)? To overcome this issue, we propose the following two-step approach:

- Step 1: **referencing phase**. In this phase we create reference lanelets that contain all ID information and a bounding box around each lane lines. The bounding box approach allows a certain deviation from the reference line but still ensures accurate mapping of the new meausurement to the existing lanes. The source of the reference lanes is discussed later.
- Step 2: **accurate phase**. In this phase we read new measurements from the raw files, and match the lanes in it to the reference lines. Once this is done, the stored reference line is recalculated and made more accurate with the information held by the new measurement. The assumption is the more occurance of one lane line the more accurate the reference line.

The two-phase approach is also illustrated in Figure 17. The reference file can come from three different sources. When the route is not yet included in the CRP map database, the Open Street Map database can be used as a prior information or own

measurements can be used. In the latter case, manual labelling is necessary, and shall be included e.g., in the file name. This way, a prior reference line can be used, based on which the boundary boxes are calculated. If the route is included in the CRP database already, it can be read in and used for the accurate phase. In all cases, creation of the bounding boxes is necessary. For this, details are provided later.
In the accurate phase the new meausurement is associated with the bounding boxes. Then, the existing reference lines and the new measurement are merged, and regression is applied on them to yield the new, more accurate reference line. Finally, the database is updated by feeding the more accurate reference line structures back.

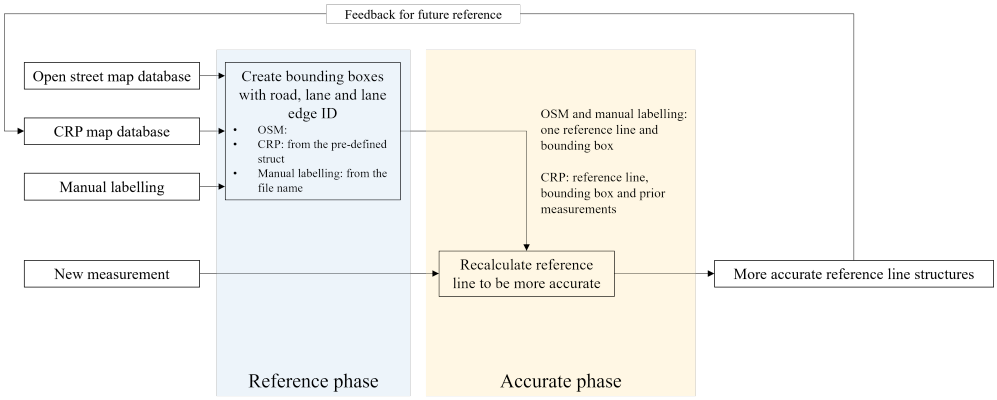


**Fig. 17**: Two phase approach of reference line creation.

## 9.3 Manual Reference Phase

Manual reference lines can be calculated by using own measurements and labelled road and lane id information.

Note: the steps given in this section are implemented in *createManualReference.m*

### 9.3.1 Strictly Monotonic Increasing Coordinates

The goal is to rotate sections of the route into a frame which provided strictly monotonic increasing X values. This helps to formulate the regression problem as a function approximation where the $X_{rotated}$ coordinates are the independent variables. Then, a selected function can be fitted. Even though this is a very simple situation, the following procedure can be used in all similar cases, where straight sections are connected via curves:

- calculate curvature along the route and cut the route per straights and curves,
- rotate snippets to have strictly monotonic X values,
- regression of polyline onto the data.

Without analytically fitting a curve a telling the curvature based on the curve parameters, numerical derivation can be used. Nominally, the curvature is the rate of the orientation change, where orientation stands for the tangential direction angle, the rate is calculated based on the step along the $X$ axis. In the end, the curvature will be given in $\frac{1}{m}$ unit given the $X$ is in $meters$. The curvature can be calculated per (3).

$$\kappa(t) = \frac{d\Theta}{dX} = \frac{d^2Y}{dX^2} \tag{3}$$

However, the route is given in discrete time, sampled with $T = T_s$ step times. Therefore, (3) can be rewritten to numerical derivations:

$$\Theta[k] = \frac{Y[k] - Y[k-1]}{X[k] - X[k-1]} \tag{4}$$

$$\kappa[k] = \frac{\Theta[k] - \Theta[k-1]}{X[k] - X[k-1]} \tag{5}$$

Assuming that $X[k] \neq X[k-1]$. However, in the example shown in Figure 20, this condition is not fulfilled necessarily. Therefore, a pre-check and transformation are needed, to ensure that $X$ gets strictly monotonic. Algorithm steps are the following:

1. Step 1: calculate mean orientation of the complete route (i.e., $\overline{\Theta} = \frac{1}{N}\sum_{k=1}^{N}\Theta[k]$)
2. Step 2: rotate the route with $\overline{\Theta}$ and translate it to point $\begin{bmatrix}0 & 0\end{bmatrix}$
3. Step 3: check, if $X$ is strictly monotonic, if yes, end of algorithm, if no, then continue with Step 4,
4. Step 4: iterate from the first point along the $X$ axis, until $\left|X[k] - X[k-1]\right| \geq dX_{min}$, if not, then Step 5,
5. Step 5: insert braking point, store previously checked $X - Y$ pairs, reinitialize the checking by repeating Step 3 and Step 4, now with the points deduced by the stored points. Iterate Step 3-5 until the end of the route.

After this algorithm, there are pre-checked snippets, all with strictly monotonic $X$ values, therefore (4-5) can be calculated for all of them. Then, the calculated curvature vectors are concatenated, resulting in the curvature of the original, uncut point series. Now, the following check is done:

1. Step 1: find sections, where $|\kappa| < \kappa_{min}$ and label them as straights - $type = 0$,
2. Step 2: find sections, where $\kappa < -\kappa_{min}$ and label them as right curves - $type = 2$,
3. Step 3: find sections, where $\kappa > \kappa_{min}$ and label them as left curves - $type = 1$.

Now, given the curve information, the route can be cut into snippets.

| Note: the steps given in this subsection are implemented in *calculateRoadCurvatureTypes.m* |
|---|

### 9.3.2 Bounding Box Creation

Now, taking the snippets for each reference lanes, a bounding box is calculated which forms a band around the reference line. By using a band, a certain deviation from

the exact reference line is allowed. This is necessary, as the new measurements must be matched with the reference lines.

Note: the steps given in this subsection are implemented in *createBoundingBoxes.m*

## 9.4 Accurate Phase

The accurate phase reads in the reference file (from any source), maps the new measurements to the reference lines and make the regression again to yield more accurate reference lines.

Note: the steps given in this subsection are implemented in *createMap.m*

### 9.4.1 Regression

All measurements plotted in the UTM coordinate frame is seen in Figure 18. The circles indicate the starting point of each lines. A few anomalies are seen:

- not all the measurements start at the same localization, there are significant differences for some of them,
- there is a deviation between estimated lines due to the aformentioned conditions. These errors are more illustrated in Figure 19.

Theoretically, the variance of the lane edge estimates can be calculated. However, there are two major issues:

- there are not enough measurements to create a probability distribution from point to point,
- the points are not sampled at the same position,
- orientation of the lane changes between $-\Pi$ and $\Pi$, which may lead to non-monotonic X values, therefore function interpretation is difficult.

The second and third problem are illustrated in Figure 20. As during reference phase the snippets are cut to have strictly monotonic $X$ local coordinates, the regression problem can be treated as a function approximation problem. For example, polylines may be used to describe the lane edge lines. Then, using Least-Square Regression the polyline coefficients can be calculated knowing the line data ahead. Please be noted that similar solution may be applied when data is not available preliminary (i.e., online estimation), but other techniques, such as system identification techniques must be introduced. This increases the complexity of the algorithm and reduced the accuracy. In this case the data is available, therefore a full regression can be applied offline, in MATLAB, using preliminary function called *polyfit(x,y,n)* of MATLAB, where $x$ and $y$ are vectors of the measurement points, and $n$ is the degree of the polyline. The most important is parameter of the regression is $n$. Choosing a low $n$ saves computational time but may lead to poor fit on the data while large value of $n$ results in good fit accuracy but takes more time to calculate and is prone to overfit on the data, therefore extrapolation is difficult. The problem can be simplified if the shape of the line to be estimated is simpler, which can be achieved but cutting the

complete route to snippets. In this case, the route consists of the clearly seen straight section followed by a right curve.

| Note: the steps given in this subsection are implemented in *calculateSmoothLine.m* |
|---|

### 9.5 Results

The results are shown in Figure 21. This is an example, where the original measurement points are shown with small dots, while the regression line is shown as solid.

## References


[1] Gmbh., R.B.: Chassis Systems Control Second generation multi purpose camera (MPC2). [Online; accessed 19-February-2024] (2013). "http://auto2015.bosch.com.cn/ebrochures2015/automated/cc/da/mpc2/datenblatt_mpc2_en.pdf"

[2] Sensors, G., Solution, N.: User Manual ADMA-G, ADMA-Speed, ADMA-Slim. https://genesys-offenburg.de/support/guides-and-manuals/adma/adma-manual/, Offenburg, Germany (2019)

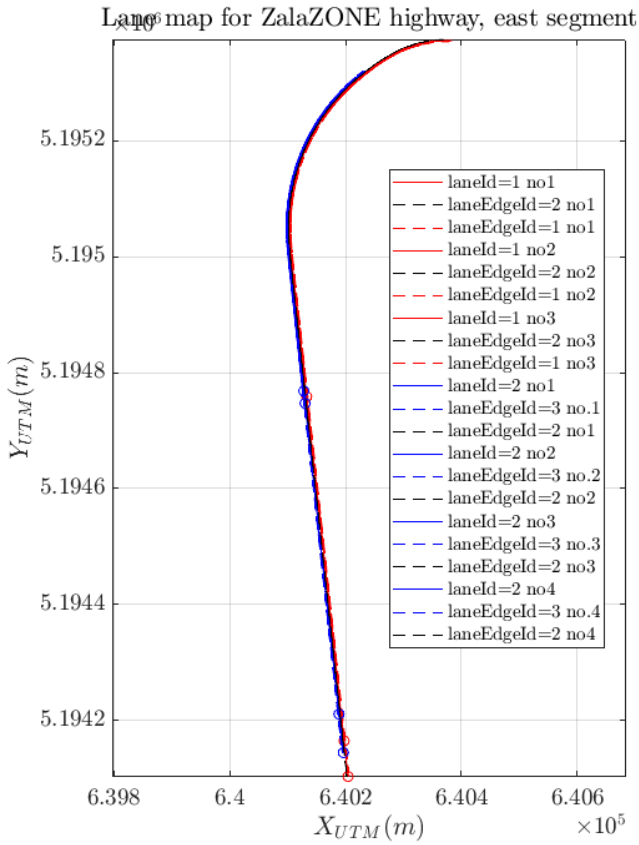


**Fig. 18**: Result of the map data preprocess - sorted lanes from multiple measurement file.

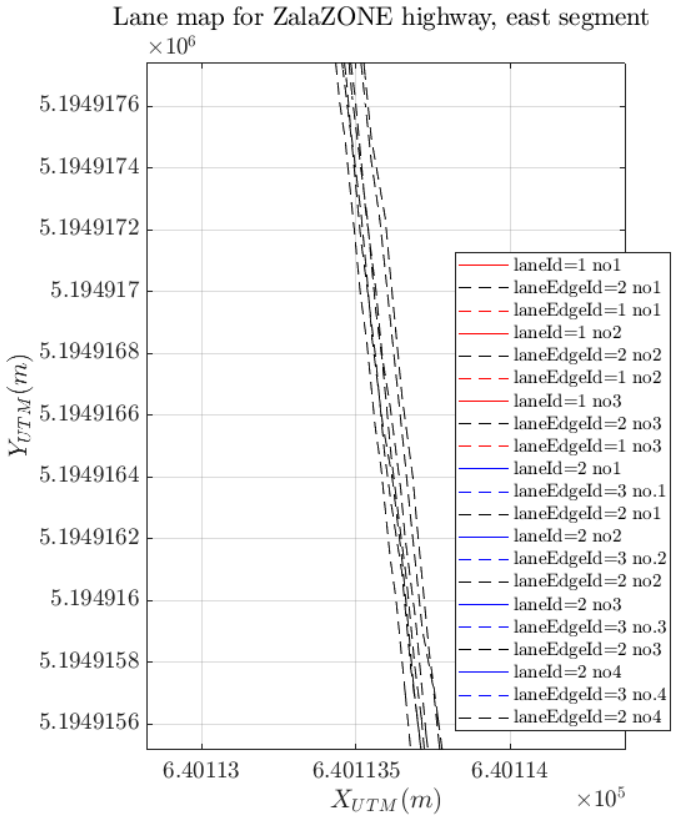


**Fig. 19**: Lane edge id=2. There is a deviation between estimates coming from multiple measurements.

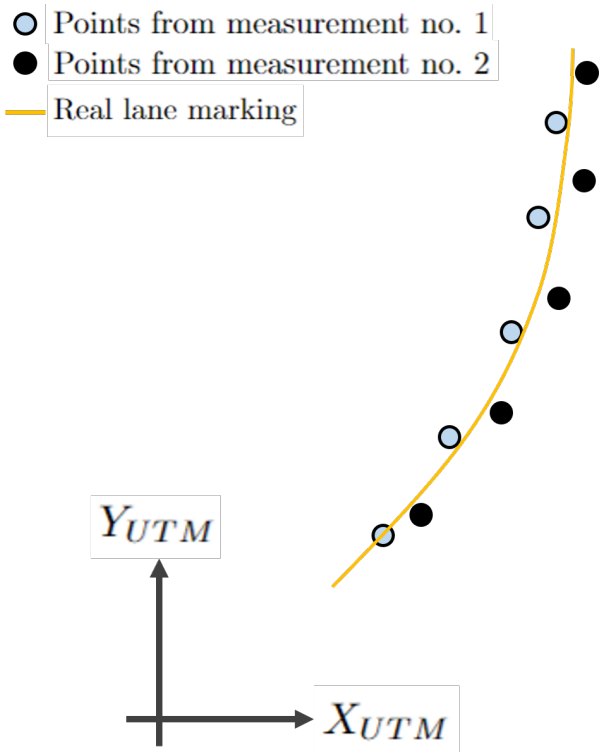


**Fig. 20**: Illustration of a realistc lane marking detection scenario. Problems are that a) there is a deviation of points due to the localization and detection inaccuracies, and b) the points are given in equal time steps, but without synchronization to each other (measurements are recorded after each other).

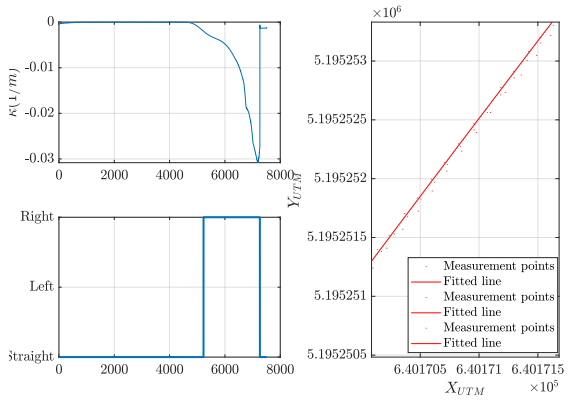


**Fig. 21**: An example (lane id 1 midlane) of the fitting result. Zoomed in to show the regression performance and make the original measurement points visible.

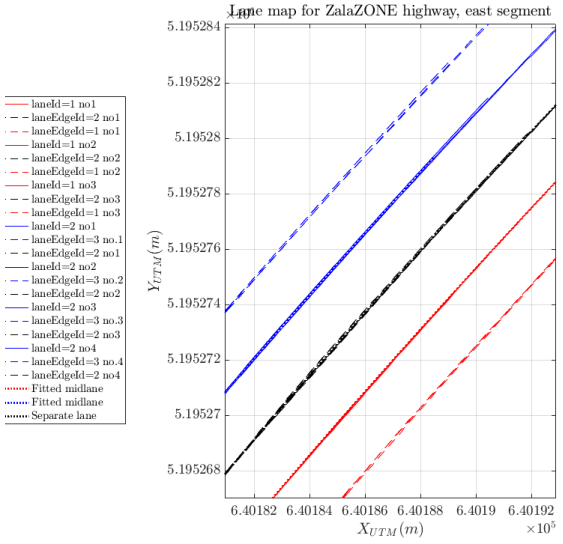


**Fig. 22**: Three lines are fitted, and visualized together with the original figure.