



Audi Autonomous Driving Cup 2018 Software Description

Current Version:	1.1
Created:	13 June 2018
Last changed:	03 July 2018
Valid for:	AADC Software Package 1.0.2

Change Documentation

Title: AADC Software Description 2018

Version: 1.1

Description	Author	Date	Version
First Release within new document	denkfl	06 June 2018	1.0
Update wheel speed controller documentation, Moved position filters to AADC Base, Gitlab repo, UserConfiguration project	denkfl, heinpe	03 July 2018	1.1

List of Figures:

Figure 1 Clonezilla Start Menu	8
Figure 2 Clonezilla Language Selection	9
Figure 3 Sample OpenDRIVE Map with multiple road sections and render of extracted path data... 15	
Figure 4 Sample OpenDRIVE Map with multiple road connections.	17
Figure 5 Sample track	18
Figure 6 Arduino EgoMotion Sensor	19
Figure 7 Arduino Ultrasonic Sensor	20
Figure 8 Arduino Battery Sensor	21
Figure 9 Arduino Actuator	22
Figure 10 Basler Camera Plugin	26
Figure 11 Laser Scanner	27
Figure 12 Signal Value Generator Plugin	28
Figure 13 Universal Camera	28
Figure 14 Calibration Plugin	29
Figure 15 Calibration XML Plugin	30
Figure 16 Camera Calibration	32
Figure 17 GUI Camera Calibration	33
Figure 18 Car Controller Plugin	34
Figure 19 Car Controller Lights	35
Figure 20 GUI Car Controller Slider	35
Figure 21 Converter Wheels Plugin	36
Figure 22 Driver Module Plugin	37
Figure 23 GUI Driver Module	38
Figure 24 Fisheye Undistortion Plugin	39
Figure 25 Lane Detection	40
Figure 26 Lane detection filter output showing the line points found on the detection lines within the defined region of interest and with the predefined boundaries for the line width.	41
Figure 27 Map Visualization Plugin	25
Figure 28 Marker Detector Plugin	43
Figure 29 Marker Positioning Plugin	23
Figure 30 Visualization of parking slot positions and numbering	24
Figure 31 Object Detection Filter	47
Figure 32 Sensor Visualization Plugin	48
Figure 33 Wheel Speed Controller Plugin	49
Figure 35 OpenCV Template Filter	50
Figure 36 User Template Filter	51
Figure 37 default_streaming_graph	53
Figure 38 default_filter_graph	54
Figure 39 Running session view	54
Figure 40 recording_graph filter graph	55
Figure 41 playback_graph streaming graph	55
Figure 42 default_filter_graph	56
Figure 43 devel_streaming_graph	56
Figure 44 devel_filter_graph	57
Figure 45 Running session view	57
Figure 46 camera_calib_graph streaming graph	58

Figure 47 camera_calib filter graph	58
Figure 48 Running session view	59
Figure 49 GUI Jury Application	63

Content

1	Linux	8
1.1	Flashing a new image	8
1.2	WLAN Credentials	10
2	ADTF.....	11
2.1	ADTF License	11
2.2	ADTF Documentation.....	11
2.3	AADC ADTF Source Package.....	11
2.3.1	Update the Source Package	12
2.3.2	Building Plugins	12
2.3.3	Development under Windows	12
3	Preinstalled SDKs	13
3.1	QT.....	13
3.2	OpenCV with contrib-modules.....	13
3.3	cuDNN	13
3.4	Pylon	13
3.5	Cuda	13
3.6	RPLidar	13
3.7	Boost	13
3.8	Tensorflow	14
3.9	gtest.....	14
4	Base Software for Visualization on a Map	15
4.1	Available Maps	15
4.2	Coordinate Systems	17
4.3	Positioning system configuration	17
5	AADC Plugin Documentation.....	19
5.1	AADC Base.....	19
5.1.1	Streaming Filter	19
5.1.1.1	Arduino EgoMotion Sensor	19
5.1.1.2	Arduino Ultrasonic Sensor.....	20
5.1.1.3	Arduino Battery Sensor	21
5.1.1.4	Arduino Actuator	22

5.1.2	Filter.....	23
5.1.2.1	Marker Positioning	23
5.1.2.2	Map Visualization.....	25
5.2	AADC Demo.....	26
5.2.1	Streaming Sources	26
5.2.1.1	Basler Camera	26
5.2.1.2	Laser Scanner	27
5.2.1.3	Signal Value Generator	28
5.2.1.4	Universal Camera	28
5.2.2	Filter.....	29
5.2.2.1	Calibration	29
5.2.2.2	Calibration XML.....	30
5.2.2.3	Camera Calibration	32
5.2.2.4	Car Controller	34
5.2.2.5	Converter Wheels.....	36
5.2.2.6	Driver Module.....	37
5.2.2.7	Fisheye Undistortion	39
5.2.2.8	Lane Detection.....	40
5.2.2.9	Marker Detector	43
5.2.2.10	Object Detection	47
5.2.2.11	Sensor Visualization	48
5.2.2.12	Wheel Speed Controller	49
5.3	AADC User	50
5.3.1	OpenCV Template Filter	50
5.3.2	User Template Plugin	51
5.4	How to program your first plugin	51
6	ADTF Projects and Sessions.....	53
6.1	Live Visualization Project	53
6.1.1	default_session	53
6.1.1.1	default_streaming_graph.....	53
6.1.1.2	default_filter_graph.....	54
6.1.2	recording	54

6.1.2.1	recording_graph.....	55
6.1.3	playback.....	55
6.1.3.1	playback_graph.....	55
6.1.3.2	default_filter_graph.....	56
6.1.4	devel.....	56
6.1.4.1	devel_streaming_graph.....	56
6.1.4.2	devel_filter_graph.....	57
6.1.5	camera_calibration	57
6.1.5.1	camera_calib_graph.....	58
6.1.5.2	camera_calib	58
6.1.6	default_session_win	59
6.2	User Configuration Project	59
6.2.1	default_session	59
6.2.1.1	default_streaming_graph.....	60
6.2.1.2	default_filter_graph.....	60
7	Jury Communication	62
7.1	General Procedure	62
7.1.1	Jury Application	62
7.1.2	Jury Actions	63
7.1.3	Car States	63
7.1.4	Sectors and Maneuvers	64
7.1.5	Maneuverlist	64
7.2	Example for Testing Jury Communication	65
7.2.1	Starting the car	65
7.2.2	Stopping the car	70
7.2.3	Finishing the course.....	70

1 Linux

- Kernel Version: 4.4.0-127
- GCC Version: 5.4.0
- Distribution: XUbuntu 16.04.2 LTS
- User Name: aadc
- User Password: aadc2018

1.1 Flashing a new image

1. Backup our ADTF license file first! Please see also `/home/aadc/License/*.lic`
2. Download and create Clonezilla bootable USB Drive
<http://clonezilla.org/liveusb.php>
3. Prepare USB Stick with Linux Image from <https://www.audi-autonomous-driving-cup.com/>.
Download and extract archive with tar:

```
$ tar -x AADC_Image_v*.tar
```
4. Copy all containing files to a folder on a second USB Drive
5. Making PC booting from USB Stick if not done automatically
 - a. Plug in Clonezilla USB Stick.
 - b. Press F12 while turning PC on to enter the Boot-selection menu.
 - c. Search for the Clonezilla USB Stick and select it **not** using UEFI mode.
6. Clonezilla menu will be started.



Figure 1 Clonezilla Start Menu

7. Select one of the first three entries, i.e. “Clonezilla live” (Default settings, VGA 1024x768)
8. Clonezilla starts and blue/grey menu should appear

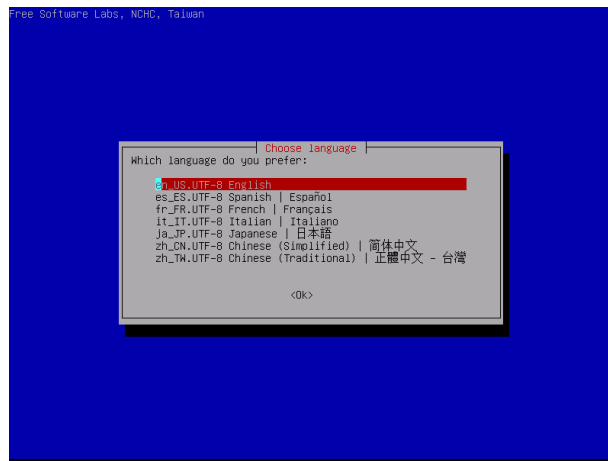


Figure 2 Clonezilla Language Selection

9. Select Language
10. Modify keymap if necessary
11. Select Start_Clonezilla
12. Select device_image
13. Select local_dev
14. Plug second USB Stick with image in the board
15. Press Ctrl-C if the the USB Device with the image to be restored is listed correctly
16. Select the second USB Stick with the image in the menu (in most times the last entry if it was plugged in at the correct time)
17. Select the parent folder with the image
18. Some file system usage infos are shown, press Enter to continue
19. Select Beginner
20. Select restoredisk (if restoredisk is not available no suitable image was found in the folder selected in 17.)
21. Next screen should show the image file, press Enter to continue
22. Choose the target (only "nvme0n1_128GB_SAMSUNG..." as the local PCIE m.2 drive should be shown)
23. Confirm the checking of image with Enter
24. Press "Enter" to continue
25. Press "Enter" again
26. Press "y" to accept
27. Press "y" again
28. Restoring should be started now
29. Now drink a coffee or look for some interesting news on the internet
30. If restoring was finished press "Enter"
31. Select poweroff
32. Remove all USB Disks
33. Switch PC on
34. If necessary, press "F" in blue XUbuntu screen to fix some errors automatically
35. Maybe you will be asked which panel config to use after starting. Select "Use Default Config"

1.2 WLAN Credentials

Each team gets one WLAN Access Point TP-Link Nano Router.

The WLAN SSID and its corresponding password and the system logins are printed on the delivered Wireless Router.

2 ADTF

2.1 ADTF License

Each vehicle is delivered with one valid ADTF License file. It is located in

```
/home/aadc/License/
```

2.2 ADTF Documentation

The ADTF Documentation is located at:

```
/opt/ADTF/3.3.1/doc/ADTF_Documentation.html
```

or

https://support.digitalwerk.net/adtf3_guides/index.html

2.3 AADC ADTF Source Package

The vehicles are delivered with a Source Package which helps the teams getting started with their own ADTF filter development. It provides a lot of standard functionalities to use the vehicle, meaning how to read its sensors, control its actuators and understand the conditions of the competition.

The Source Package contains many different ADTF Filters in three categories:

- AADC Base Filters
- AADC Demo Filters
- AADC User Filters

The filters in AADC Base serve their users mainly as the standard communication with the Arduinos. The teams must not modify these filters. If any question or errors occur, refer to the website forum and the described procedure in the regulations. The Base Filter are described in 5.1

The filters in AADC Demo offer an extended scope of functionalities for the use of the vehicle. The package contains filters to visualize sensor values, to do calibration procedures, to convert values or several ways to control the speed and steering controllers of the car. All these filters can be modified by the teams or can be used as a start-up for their own algorithms and implementations. The Demo Filters are described in 5.2

The AADC User Filter package contains one template filter without any specific functions. All the individual filters of the teams have to be placed in this package and its corresponding directory. The User Filter are described in 5.3.

The Source Package also contains some ADTF Configurations, which explain the usage of the compiled filters. The ADTF Configuration is described in 6.

The Source Package works on both Linux and Windows. For developing on Windows, further ADTF Licenses have to be requested (chapter 2.3.3).

2.3.1 Update the Source Package

The latest releases of the Source package will be published in the following public GitLab repository <https://git.digitalwerk.net/aadc/adtf-basic-software>. Additionally every SW release will be announced via the AADC homepage forum.

2.3.2 Building Plugins

All the three categories in the source package have a prepared build environment including a complete CMake-Project.

The user has to execute the corresponding script to build the sources.

The delivered vehicle contains the latest source package in the folder:

```
/home/aadc/AADC
```

The source code is located in:

```
/home/aadc/AADC/src
```

In the main folder are three build scripts each for Windows and for Linux.

- Windows:

```
build_base_win.bat
build_demo_win.bat
build_user_win.bat
```

- Linux:

```
build_base.sh
build_demo.sh
build_user.sh
```

On Windows, these scripts create a build in form of a Visual Studio projects that can be opened and compiled using Visual Studio. On Linux, these scripts create a CMake Project and the gcc is used to compile the project. To use the CMake Project in another IDE the CMake-Files and especially the Generator has to be adjusted.

For development it is recommended to choose your own development IDE (CodeBlocks, Eclipse...) and load the CMake projects there.

A lot of useful information can be found also in the ADTF SDK. Search for some of following topics:

- ADTF CMake Introduction
- Using CMake to build the ADTF Examples
- Programming ADTF
- ADTF Plugin SDK - Developing my first ADTF plugin

2.3.3 Development under Windows

Licenses for ADTF3 are assigned by Digitalwerk GmbH with the following process:

1. Register at the Digitalwerk Support Portal: <https://support.digitalwerk.net>
Company Registration Key (Academic/Students): **5pq-w56-j3y**
2. Send a request for an ADTF3 license to support@digitalwerk.net with reference to the AADC and attach the MAC address(es) of the devices on which you intend to use ADTF3.
3. You receive the license file(s) asap.

3 Preinstalled SDKs

3.1 QT

- Version: 5.9.0 (required by ADTF)
- Path: /opt/qt/5.9.0
- Download <http://download.qt.io/archive/qt/5.9/5.9.0/>
- License LGPL License

3.2 OpenCV with contrib-modules

- Version: 3.4.1
- Path: /opt/opencv/3.4.1
- Download <http://opencv.org/downloads.html>
- License BSD License

3.3 cuDNN

- Version: 7.1.
- Path: /opt/cudNN/7.1.
- Download <https://developer.nvidia.com/cudnn>
- License NVIDIA cuDNN License Agreement

3.4 Pylon

- Version: 5.0.12
- Path: /opt/pylon/5.0.12
- Download <https://www.baslerweb.com/de/produkte/software/?gclid=C00iudz039MCFdXNGwodlBcDqw>
- License Basler SOFTWARE LICENSE AGREEMENT

3.5 Cuda

- Version: 9.2
- Path: /opt/cuda/9.2
- Download <https://developer.nvidia.com/cuda-toolkit>
- License Nvidia Software License Agreement

3.6 RPLidar

- Version: 1.6.0
- Path: /opt/rplidar/1.6.0
- Download <https://www.slamtec.com/en/Support#rplidar-a3>

3.7 Boost

- Version: 1.66.0
- Path: /opt/boost/1.66.0
- Download https://www.boost.org/users/history/version_1_66_0.html
- License: <https://www.boost.org/users/license.html>

3.8 **Tensorflow**

- Version: 1.7.0
- Path: /opt/tensorflow/1.7.0
- Download <https://github.com/tensorflow/tensorflow/releases/tag/v1.7.0>
- License: <https://github.com/tensorflow/tensorflow/blob/master/LICENSE>

3.9 **gtest**

- Version: 1.8.0
- Path: /opt/gtest/1.8.0
- Download <https://github.com/google/googletest/tree/release-1.8.0>
- License: <https://github.com/google/googletest/blob/master/googletest/LICENSE>

4 Base Software for Visualization on a Map

4.1 Available Maps

OpenDRIVE is an open file format for the logical description of the road networks. OpenDRIVE has an XML-based syntax which divides the road network into a number of road sections. The map data can contain any number of road sections but should have at least one road section.

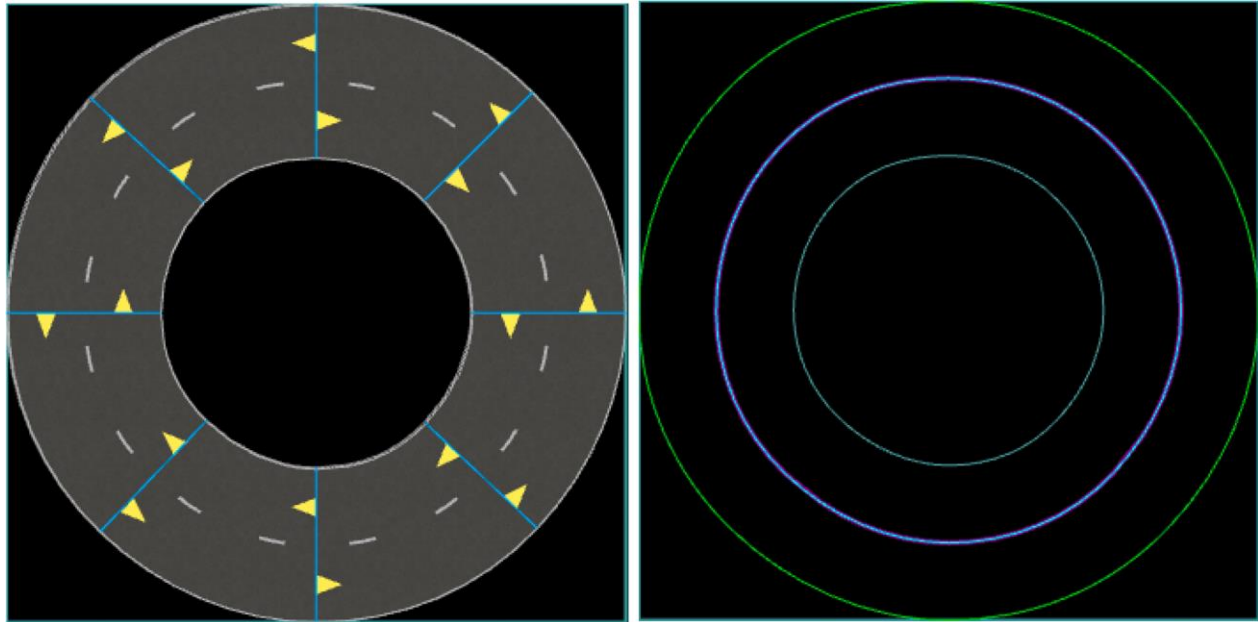


Figure 3 Sample OpenDRIVE Map with multiple road sections and render of extracted path data.

Each road section has information about the geometry and profile of the road, number of lanes in the road and connections to other roads etc. Detailed specification of OpenDRIVE format can be found in www.opendrive.org. The following paragraph describes the XML-tags that are relevant for extracting path profiles for route planning.

An XML-parser strips data from the OpenDRIVE file. Attribute values of the XML data provide the information about the roads. Road sections provide the profile of the path as parametric cubic polynomial with parameter p in the range $[0;1]$. Any number of finite points can be created on the path of the road by choosing the right point interval. This interval can be different for each road element based on the length of the road section. To proceed further all the states x and the list of neighbor states $U(x)$ for each state must be extracted from the OpenDRIVE map. The algorithm used for extracting states and connections is as follows:

- 1: **for road do**
- 2: From <road> Get *roadId*, *junctionId*
- 3: From <link> Get Successor and Predecessor Id, *contactPoint*

```

4:          From <geometry> Get x, y, heading
5:          From <paramPoly3> Get  $aU, aV, bU, bV, cU, cV, dU, dV$ 
6:      end for
7:      for road do
8:          Find points in local coordinate  $u_{local}$  and  $v_{local}$ 
9:          Transform points to global  $x_{global}$  and  $y_{global}$ 
10:         Create Unique id i
11:         Find neighbour states  $x_i^o$ 
12:         Add  $x_{global}, y_{global}$  and  $x_i^o$  to state space X
13:     end for

```

Steps 1-6 in algorithm extract information about each road. Profile of the road is specified by parametric cubic polynomial in a local frame. Steps 7 and 8 use the extracted polynomial parameters to create map points in global frame. Global frame is the frame at the origin of the map. Equations used for extracting map points in local frame from polynomial are given below. The parameter p has values between 0 and 1; p=0 gives the location of first point and p=1 provides the location of last point in the local frame.

$$\begin{aligned}
 u_{local} &= a_u + b_u \cdot p + c_u \cdot p^2 + d_u \cdot p^3 \\
 v_{local} &= a_v + b_v \cdot p + c_v \cdot p^2 + d_v \cdot p^3
 \end{aligned}$$

Translation and rotation of local frame is provided by the parameters x, y and heading. Location of each point in the global frame is found using rotation and translation:

$$\begin{aligned}
 x_{global} &= u_{local} \cdot \cos(\text{heading}) - v_{local} \cdot \sin(\text{heading}) + x \\
 y_{global} &= u_{local} \cdot \sin(\text{heading}) + v_{local} \cdot \cos(\text{heading}) + y
 \end{aligned}$$

Each polynomial parameter p can have a unique number $m=1:M$ steps where M is the number of points for each road. This along with a unique road id provided in the map is used to create a unique global id i where $i=1,2..N$ and $N=NR \cdot M$; NR is the total number of roads. This unique id is assigned to each map point as shown in step 10 of path algorithm for identifying the next possible states from current state and creates the base for the route planning.

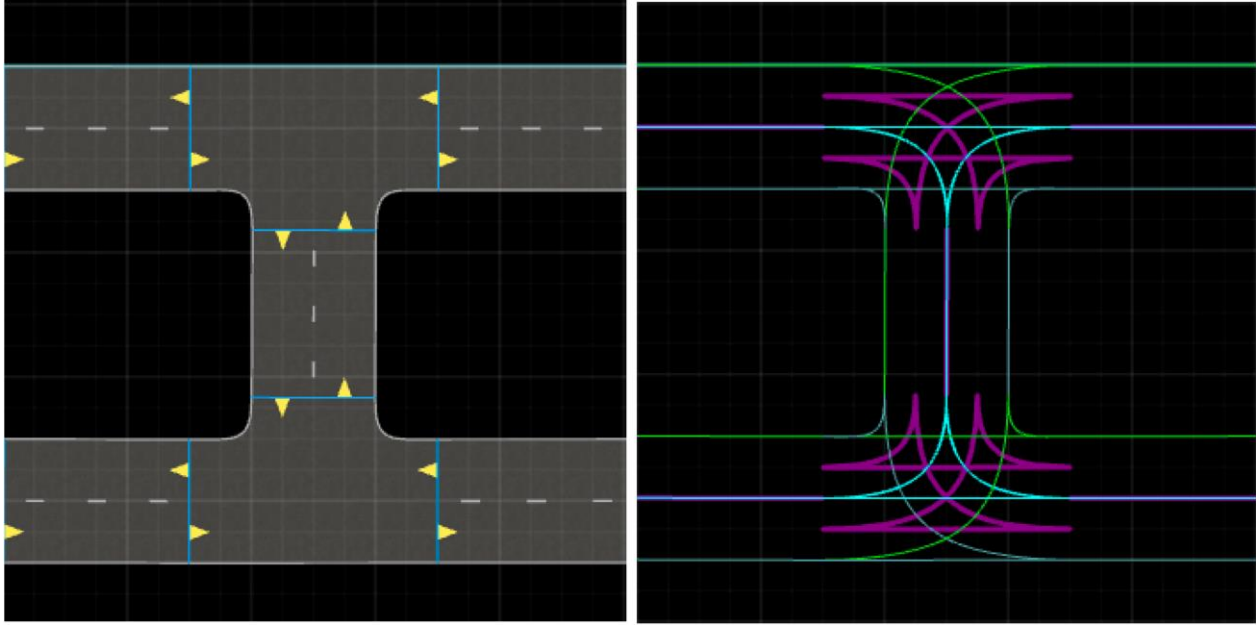


Figure 4 Sample OpenDRIVE Map with multiple road connections.

These identifiers can also be used to retrace the path once the goal is found. To find the connection between map points, connection information from roads is used. The links between roads are specified by successor and predecessor road id. The links are road ids of other roads if connection is not ambiguous. If a road has multiple successor or predecessor, then a junction id is used. A junction is nothing but a group of roads that connect neighboring roads. These group of roads together are provided by a unique junction id in OpenDRIVE maps and are used to represent multiple connections of a road. Possible next states of the current state are found based on successor and predecessor road id provided in the road information for last points ($p=0$ or 1). For other points in each road ($p \neq 0$ and $p \neq 1$), the previous or next point in the same road are the possible next states.

4.2 Coordinate Systems

OpenDrive maps are using ISO 8855 axis system definitions and thus also positioning, road sign and parking space definitions aligned accordingly.

The axis orientations for ISO 8855 has X forward, Z up, and Y pointing to the left hand side of the vehicle. The reference (Earth/OpenDrive) axis system (X_E, Y_E, Z_E) is right-handed orthogonal axis system fixed in the inertial reference. The Z_E axis being parallel to the gravity vector and pointing up.

The yaw (heading/direction) angle is then defined as an angle from X_E to X axis, about Z axis. A positive yaw value implies a left-hand rotation.

4.3 Positioning system configuration

This section describes how to setup positioning system with a given map.

The ADTF filter markerPos (5.2.14) provides a positioning system that utilizes road signs to calculate vehicle position, speed and heading. The positioning system can be configured using a road-sign XML-file. This file provides a simple mapping of the road-sign coordinates and direction of the sign. The road signs with „init“-tag are used for initializing the positioning system. These signs need to have unique IDs and the same IDs cannot be used repeatedly on the track. The positioning system is referenced locally but can be aligned with the map.

All of the provided maps have origin (0,0) in the bottom left corner. When aligning with the map the coordinates of the road signs should be calculated from the origin. Accordingly direction of the road sign should be calculated from X axis about Z axis; see 4.2.

It might not always be feasible to implement a full size track to match the particular map. In these cases it is possible to work only on a partial map. If the partial map is defined by the blue rectangular area (see the image below), the locations of the road signs are simply calculated with respect the origin of the original map. Direction (heading) calculation is handled the same way with respect to the original map.

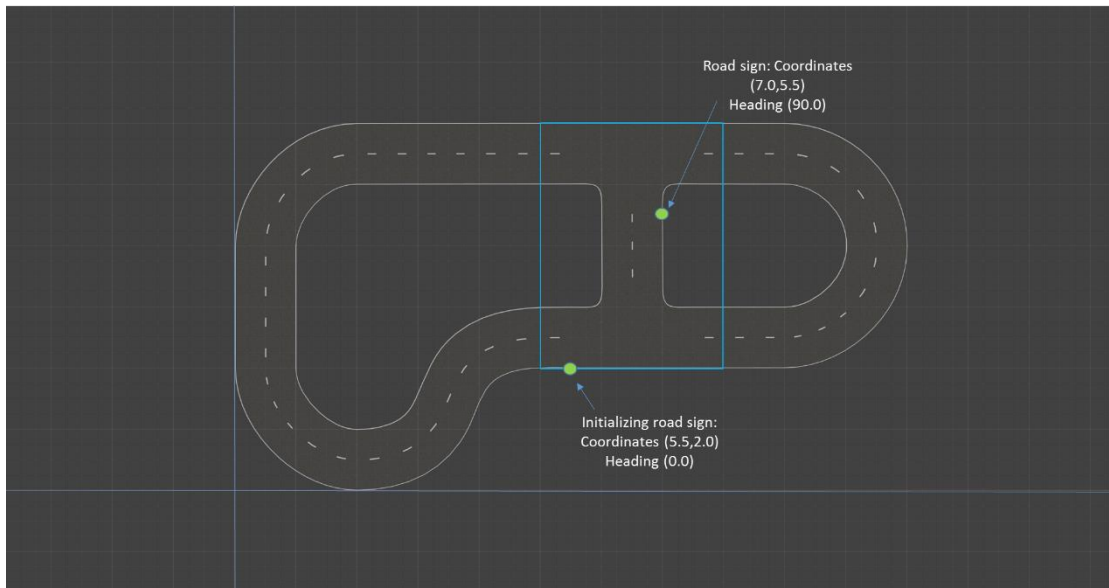


Figure 5 Sample track

Detailed description of the ADTF filter is given in 5.2.14.

5 AADC Plugin Documentation

5.1 AADC Base

5.1.1 Streaming Filter

These filters implement the communication with all Arduinos. On a Linux host the filter searches for the Arduino devices registered as /dev/ttyACM0 up to /dev/ttyACM3.

You can check them with the following command:

```
ls /dev/ttyACM*
```

Each Arduino knows which sensor it is and transmits its ID in the serial protocol. The ADTF filters try all the available ports and check the send ID of the Arduino. If the specified ID is found the serial communication is established and the transmission of the data is started. In total there should be 4 Arduinos recognized with the IDs 3, 4, 5 and 6. You can check these IDs in the `arduino_protocol.h` header if you want to know which ID belongs to which Arduino. At initialization, the Arduino frames are read but not transmitted yet. At start the filter transmits all frames to the output pins. So they could be visualized via Sensor Visualization filter. At stop the filter stops sending frames. At deinitialization the filter closes all arduino ports.

5.1.1.1 Arduino EgoMotion Sensor

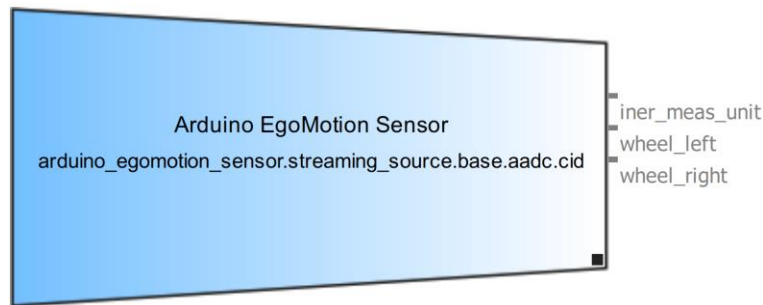


Figure 6 Arduino EgoMotion Sensor

This streaming source outputs both motion data from the inertial measurement unit and the two wheel tick encoders.

Plugin Details

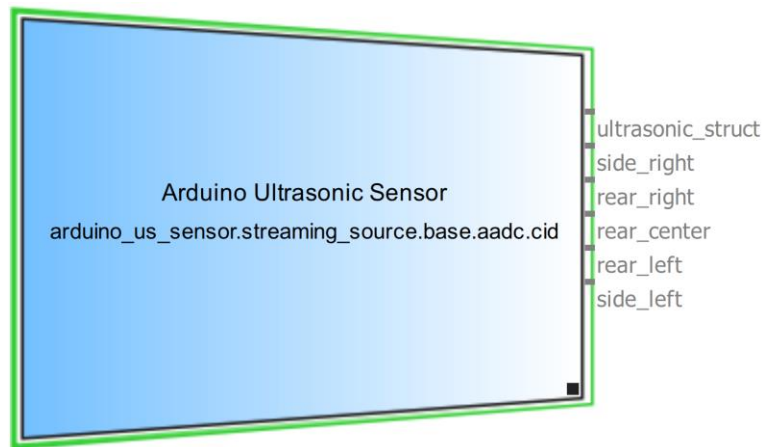
Path	src/aadcBase/arduino/ArduinoEgoMotionSensor
Filename	arduino_egomotion_sensor.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
Enable console log	Enable logging of arduino communication to console	false

Output Pins

Pin	Description	Unit
iner_meas_unit	tInerMeasUnitData from inertial measurement unit	
wheel_left	tWheelData from left wheel	count and direction
wheel_right	tWheelData from right wheel	count and direction

5.1.1.2 Arduino Ultrasonic Sensor**Figure 7 Arduino Ultrasonic Sensor**

This streaming source outputs the measurement results of the five installed ultrasonic sensors. Either each sensor via a single Pin or all values from the five sensors gathered in the `ultrasonic_struct`, which is triggered in the `rear_center` sensor.

Plugin Details

Path	src/aadcBase/arduino/ArduinoUSSensor
Filename	arduino_us_sensors.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
Enable console log	Enable logging of arduino communication to console	false

Output Pins

Pin	Description	Unit
ultrasonic_struct	tUltrasonicStruct from all ultrasonic sensors	
side_right	tSignalValue from ultrasonic sensor side right	cm
rear_right	tSignalValue from ultrasonic sensor rear right	cm

rear_center	tSignalValue from ultrasonic sensor rear center	cm
rear_left	tSignalValue from ultrasonic sensor rear left	cm
side_left	tSignalValue from ultrasonic sensor side left	cm

5.1.1.3 Arduino Battery Sensor

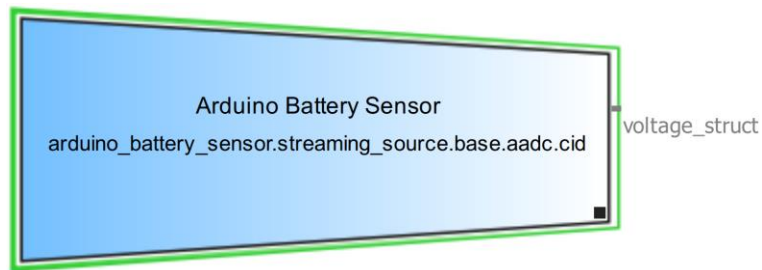


Figure 8 Arduino Battery Sensor

This streaming source outputs the measurement data from each cell of the actuator as well as the sensor battery.

Plugin Details

Path	src/aadcBase/arduino/ ArduinoBatterySensor
Filename	arduino_battery_sensor.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
Enable console log	Enable logging of arduino communication to console	false

Output Pins

Pin	Description	Unit
voltage_struct	tVoltageStruct voltage from all batteries	mV

5.1.1.4 Arduino Actuator

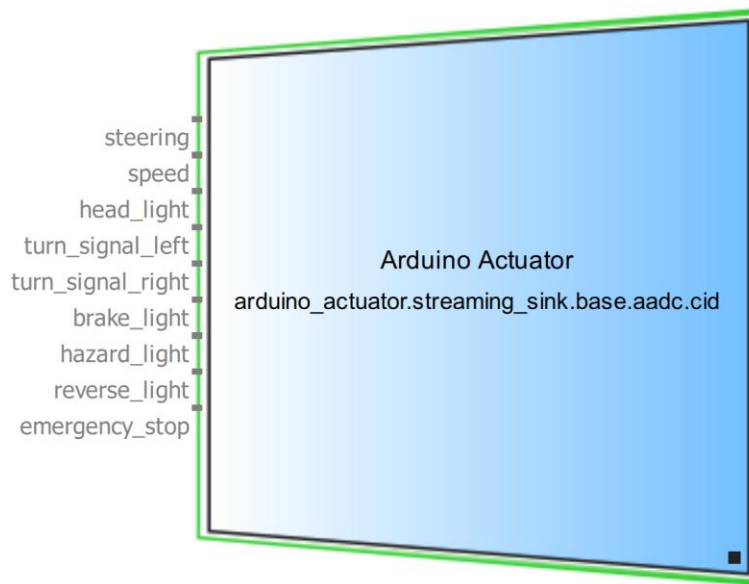


Figure 9 Arduino Actuator

Using this streaming sink you can control the actuators of the vehicle, for instance the steering and the speed as well as the lights. After startup for ten seconds zero values are transmitted for the speed, thus the driving controller needs this for its initialization routine.

Plugin Details

Path	src/aadcBase/arduino/ ArduinoActuator
Filename	arduino_actuator.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
Enable console log	Enable logging of arduino communication to console	false

Input Pins

Pin	Description	Unit
steering	tSignalValue for the steering controller	percent [-100, 100]
speed	tSignalValue for the speed controller	percent [-100, 100]
head_light	tBoolSignalValue for the headlight	boolean
turn_signal_left	tBoolSignalValue for the turn left lights	boolean
turn_signal_right	tBoolSignalValue for the turn right lights	boolean
brake_light	tBoolSignalValue for the brake	boolean

	lights	
hazard_light	tBoolSignalValue for the hazard lights	boolean
reverse_light	tBoolSignalValue for the reverse lights	boolean
emergency_stop	tJuryEmergencyStop for emergency stop	boolean

5.1.2 Filter

5.1.2.1 Marker Positioning

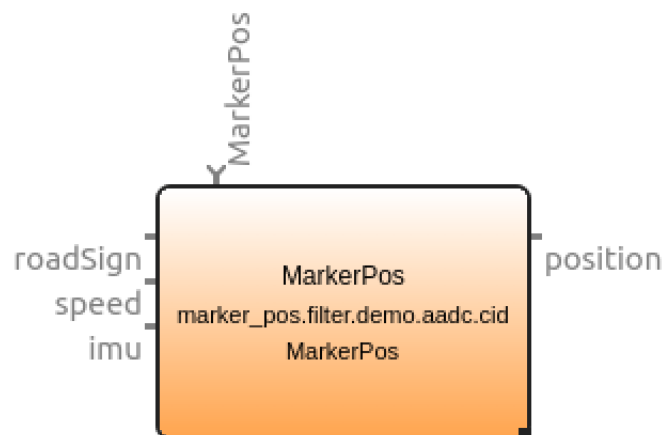


Figure 10 Marker Positioning Plugin

This filter provides positioning solution based on detected road-signs (markers), and sensor dead-reckoning. The filter implements a Kalman filter to provide sensor fusion and dead-reckoning solution when the road-signs are not visible. The state of the vehicle is represented by its location (ENU-frame), heading and speed. Additionally, the filter tracks the yaw-gyro bias and the odometer scaling as floating parameters.

The filter calculates distance and angle to the road-sign, and measures pose of the road-sign. It then uses these measurements to estimate the vehicle coordinates, speed and heading. The position solution uses the first road-sign in the configuration file to initialize the position and heading of the vehicle. This road-sign should be placed so that it is visible before the position solution is needed.

The road-sign configuration is an XML-file. It provides a simple mapping of the road-sign coordinates (ENU-frame), direction of the sign and a search radius. The coordinate system is referenced locally but should be aligned with the map data when available. The road-sign direction is the targeted vehicle heading when the road-sign is visible to the vehicle. The search radius defines the area within which the filter tries to find the sign. The search area is mainly used for marker detection integrity monitoring.

The configuration file also contains the position and ids of parking slots.

- all parking slots will be uniquely numbered
- reference positions are centered to the slots, attached to the street
- all available parking slots will be provided along with the initial traffic sign xml file.

This figure shall visualize the parking slot positions and numbering:

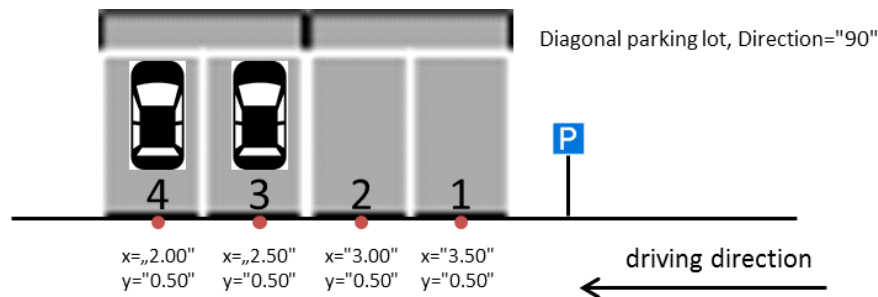


Figure 11 Visualization of parking slot positions and numbering

A sample file is shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<configuration>

  <!--first sign is used to initialize the positioning system -->

  <roadSign id="10" x="9.135" y="3.356" radius="1.0" direction="90"
init="1"/>

  <roadSign id="2" x="5.865" y="3.356" radius="1.0" direction="-90"/>

  <roadSign id="2" x="7.135" y="5.62" radius="1.0" direction="90"/>

  <roadSign id="5" x="3.644" y="0.135" radius="1.0" direction="0"/>

  <roadSign id="16" x="7.35" y="7.1" radius="1.0" direction="180"/>


  <parkingSpace id="1" x="8.26" y="1.3" status="1" direction="-90" />

  <parkingSpace id="2" x="8.73" y="1.3" status="0" direction="-90" />

  <parkingSpace id="3" x="9.26" y="1.3" status="1" direction="-90" />

  <parkingSpace id="4" x="9.73" y="1.3" status="0" direction="-90" />

</configuration>
```

Plugin Details

Path	src/aadcDemo/position/MarkerPos
Filename	marker_pos.adtfplugin

Version	1.0.0
---------	-------

Plugin Properties

Property	Description	Default
Configuration file for the road-signs	The configuration defines coordinates and direction of the road-signs that are used in the positioning solution	The first road-sign in the configuration file is used as an initialization point.
Camera Offset	The camera offset parameters can be used to compensate the camera offset in the marker detection	Default (Basler): lateral 0.0, longitudinal 0.295 The offset compensation can be defined in lateral and longitudinal direction.
Speed scale	The initial scale value for the speed measurement	Default: 1.0 This value can be adjusted if the speed scale is significantly different.

Input Pins

Pin	Description	Media Description
Speed		tSignalValue
InerMeasUnit_struct		InerMeasUnitData
RoadSign_ext		tRoadSignExt

Output Pins

Pin	Description	Media Description
Position		tPosition

5.1.2.2 Map Visualization

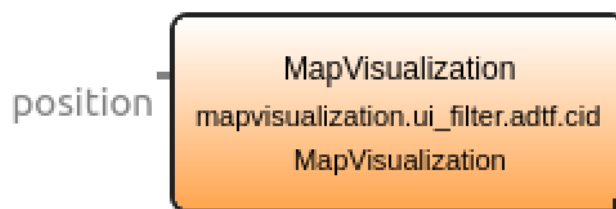


Figure 12 Map Visualization Plugin

This filter reads and renders an OpenDRIVE map, and it also displays the input position when provided.

Plugin Details

Path	src/aadcDemo/position/MapVisualization
Filename	map_visualization.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
Map File	Provides the location of Open Drive map file	Current Project Folder

Input Pins

Pin	Description	Media Description
Position		tPosition

5.2 AADC Demo

5.2.1 Streaming Sources

5.2.1.1 Basler Camera

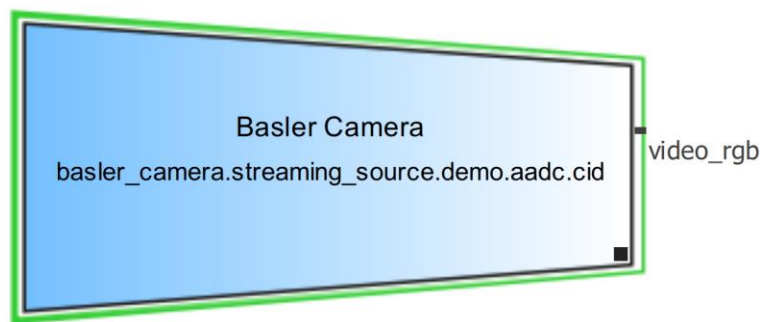


Figure 13 Basler Camera Plugin

This streaming source grabs images from the Basler Camera. The grabbing parameter for the camera can be set by in the properties of the streaming source. The grabbing procedure needs the pylon5 SDK which is provided by Basler.

Dependencies

This plugin needs the following libraries:

- Pylon5

Plugin Details

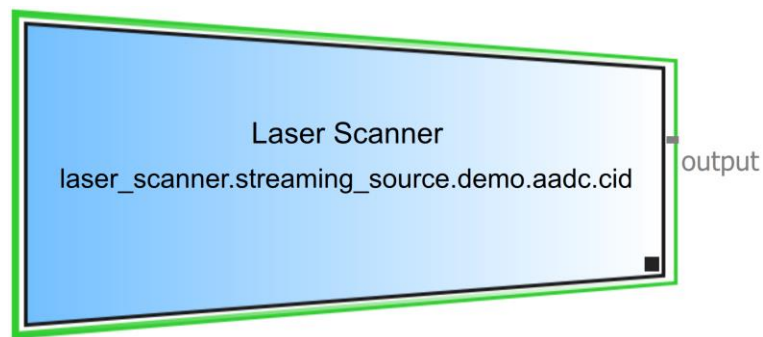
Path	src/aadcDemo/sensor/BaslerCamera
Filename	basler_camera.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
streamWidth [Pixel]	Width of the Stream	1280
streamHeight [Pixel]	Height of the Stream	960
Brightness	target Brightness for Auto Gain Function	0.3
frame_delay [ms]	Set delay between two frames	100000
ROI xOffset [Pixel]	x-Offset of the ROI	440
ROI yOffset [Pixel]	y-Offset of the ROI	330
ROI width [Pixel]	Width of the ROI	400
ROI height [Pixel]	Height of the ROI	300

Output Pins

Pin	Description	Type
video_rgb	Video output pin of camera	stream_meta_type_image

5.2.1.2 Laser Scanner**Figure 14 Laser Scanner**

This streaming source outputs laser scanner scan points for detected objects in a range from 270 degree to 90 degree whereas 0 degree is the forward direction of the vehicle. The scan points are output in polar coordinates.

Plugin Details

Path	src/aadcDemo/sensor/LaserScanner
Filename	laser_scanner.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
serial_device	The assigned serial device name	/dev/ttyUSB0

Output Pins

Pin	Description	Media Description
-----	-------------	-------------------

output	Data Struct of the Laser Scanner data	tLaserScannerData
--------	---------------------------------------	-------------------

5.2.1.3 Signal Value Generator

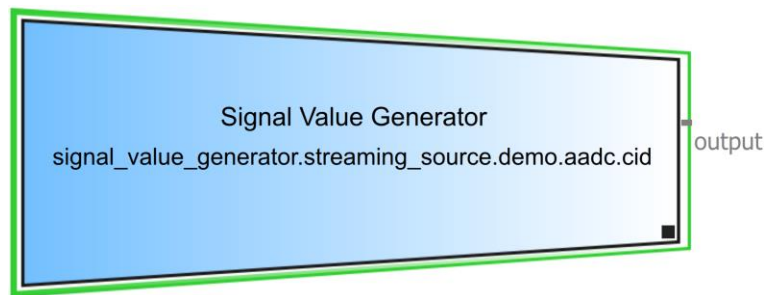


Figure 15 Signal Value Generator Plugin

This filter can be used to generate a constant value of type tSignalValue. Which can be used for example as a static input for the steering/speed controller.

Plugin Details

Path	src/aadcDemo/helper/ SignalValueGenerator
Filename	signal_value_generator.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
Timer Interval [Hz]	Defines how much updates for steering and speed controller are sent in one second (Range: 0 to 100 Hz)	30
Value	Defines the value which is transmitted	0

Output Pins

Pin	Description	Media Description
output		tSignalValue

5.2.1.4 Universal Camera

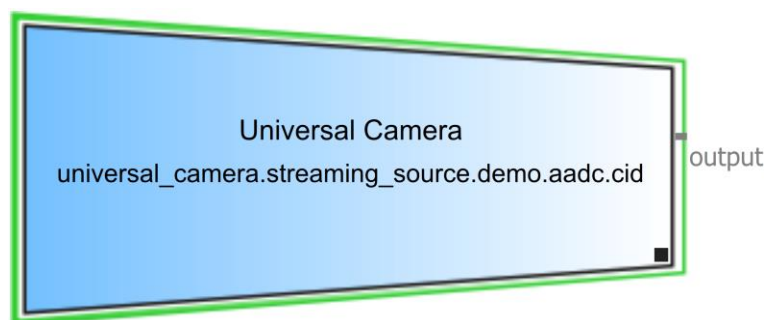


Figure 16 Universal Camera

This streaming source outputs the data from the rear view camera of the vehicle using the OpenCV `cv::VideoCapture`.

Dependencies

This plugin needs the following libraries:

- OpenCV v3.4.1

Plugin Details

Path	src/aadcDemo/sensor/UniversalCamera
Filename	universal_camera.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Used Value
ID	id of the opened video capturing device (i.e. a camera index)	0
flipHorizontal		tFalse
flipVertical		tFalse
streamChannels		3
streamHeight		960
streamWidth		1280

Output Pins

Pin	Description	Type
Output	Outputpin for RGB video	stream_meta_type_image

5.2.2 Filter

5.2.2.1 Calibration

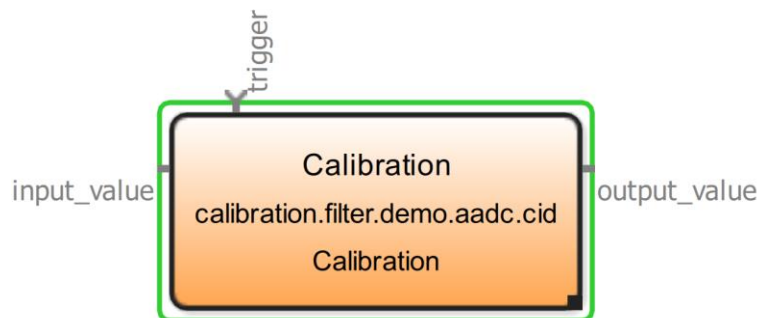


Figure 17 Calibration Plugin

This filter does a simply scaling of the value given on the input pin with the value set in the property “Scale Factor” and returns the result on the output pin.

Plugin Details

Path	src/aadcDemo/datafilter/ Calibration
Filename	calibration.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
Scale Factor	The input value is multiplied with the here given factor	1

Input Pins

Pin	Description	Media Description
input_value	input	tSignalValue

Output Pins

Pin	Description	Media Description
output_value	scaled output	tSignalValue

5.2.2.2 Calibration XML

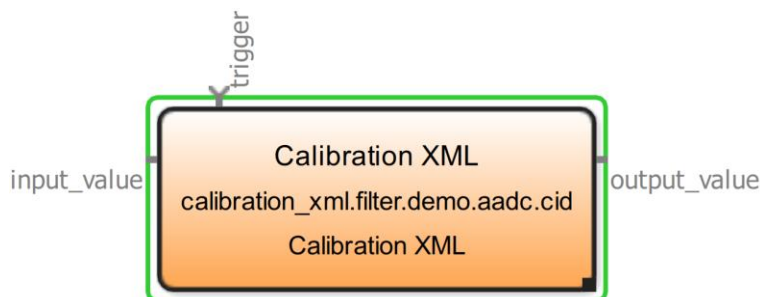


Figure 18 Calibration XML Plugin

This filter does in comparison to the Calibration Filter an extended calibration.

The user has to generate a XML-File which has to be set in the property. This XML-File must include a calibration table with x- and y-values and a mode which has to be used for interpolation.

When the filter receives an input value it looks on the x-Axis of the calibration table and gets the corresponding value on the y-Axis using the set interpolation. This result is transmitted on the pin output_value. If the value on the input pin is greater than the maximum value in the table the maximum value is used, if it is smaller than the minimum value the minimum value is used.

The x-Values in the table must be in increasing order otherwise the calibration does not work and prints an error.

The structure of the XML has to be like this:

```

<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<calibration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <settings>
    <mode>linear</mode>
  </settings>
  <supportingPoints>
    <point>
      <xValue>307</xValue>
      <yValue>-45</yValue>
    </point>
    <point>
      <xValue>572</xValue>
      <yValue>45</yValue>
    </point>
  </supportingPoints>
</calibration>

```

Plugin Details

Path	src/aadcDemo/datafilter/ CalibrationXML
Filename	calibration_xml.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
Border Warnings to Console	If enabled a warning is printed to console each time the border points of the given xml are reached	False
Configuration File for Interpolation	The XML to be loaded has to be set here	
Print initial table to Console	If enabled the loaded points of the interpolation table of the XML are printed to console	False

Input Pins

Pin	Description	Media Description
input_value		tSignalValue

Output Pins

Pin	Description	Media Description
output_value		tSignalValue

5.2.2.3 Camera Calibration

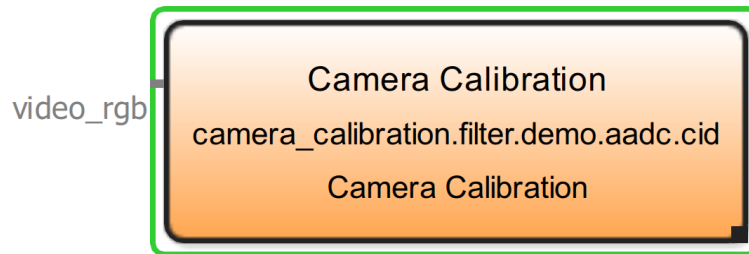


Figure 19 Camera Calibration

This filter can be used for doing the intrinsic calibration for any used camera. After starting a GUI is shown which displays the video stream and has four buttons for starting the calibration, saving the file and cancelling the calibration process. The calibration is done with the standard opencv function for intrinsic calibration described in the opencv documentation. The main functions are used from opencv samples in (opencv/samples/cpp/calibration.cpp). For further information have look in this documentation.

To perform the calibration with this ADTF Filter a calibration pattern has to be printed first. A standard chessboard pattern is located in src\aadcdemo\camera\CameraCalibration\pattern.png and has to be printed on an A3 paper. It is better to use a thick paper or stick the paper to solid background.

The side length of one square of the printed chessboard pattern has to be measured and the length be set in the property Square Size in the Filter (unit is meter).

After that the RGB Output of any camera filter has to be connected to the input pin of this filter and the ADTF configuration has to be started.

Depending on the camera connected to plugin select "Start Calibration Fisheye" or "Start Calibration"

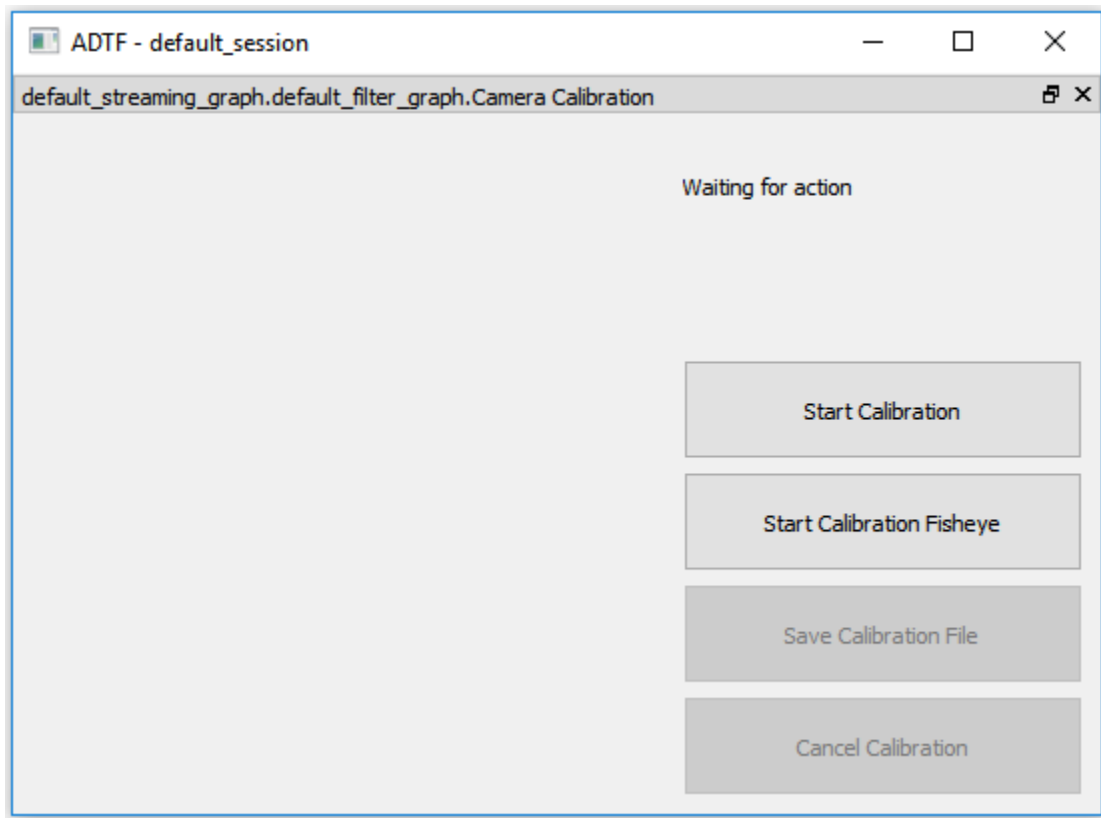


Figure 20 GUI Camera Calibration

Now the filter searches for the selected pattern in the image. If it was found it is shown in the windows and the points are saved as valid dataset. For each calibration routine, a certain number of valid data sets is necessary (can be set in the filter properties)

If the number of valid datasets are captured the button "Save Calibration File" can be pressed. The calibration will be checked and if it has a valid result it will be saved to the selected destination. If the calibration result cannot be used just have another try or modify the number of datasets or the delay between.

The resulting calibration file can be used for the plugins which need an intrinsic calibration file

Dependencies

This plugin needs the following libraries:

- QT v.5.9.0
- OpenCV v3.4.1

Plugin Details

Path	src/aadcDemo/camera/CameraCalibration
Filename	basler_camera.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
Width [number of squares]	The number of squares in horizontal axis (Range: 5 to 15)	8
Height [number of squares]	The number of squares in vertical axis (Range: 5 to 15)	6
Square Size [m]	Square size (length of one side) in meters	0.025
Aspect Ratio	Fix aspect ratio (fx/fy) (1 by default)	1
Calibration Pattern	Defines the pattern which is used for calibration	Chessboard
Number of Datasets to use	Set the number of datasets which are used for calibration	10
Delay [s]	Delay between captured datasets in seconds	1

Input Pins

Pin	Description	Type
video_rgb	video input from camera	stream_meta_type_image

5.2.2.4 Car Controller

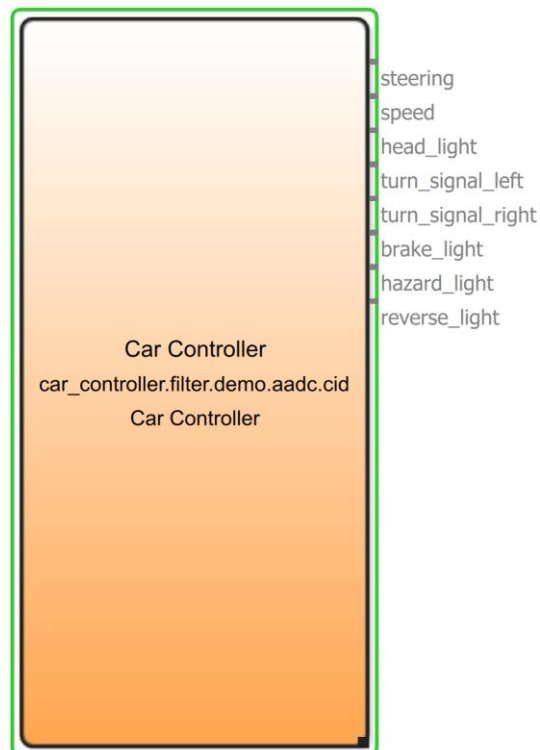


Figure 21 Car Controller Plugin

With this filter the basic functions of the car can be tested.

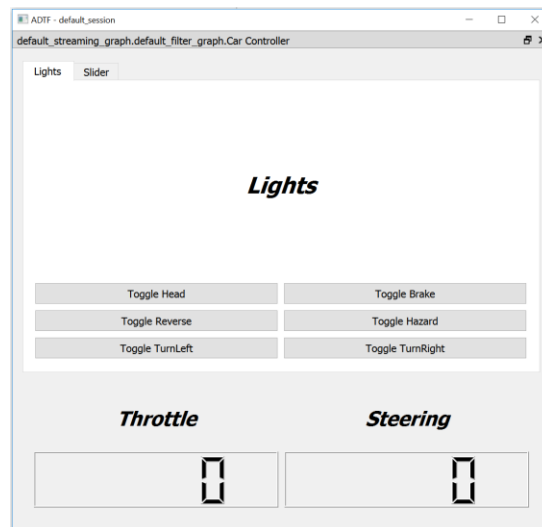


Figure 22 Car Controller Lights

This tab is for controlling the lights on the car.



Figure 23 GUI Car Controller Slider

This tab controls the steering and throttle of the car. The horizontal slider is for the steering, the vertical for the throttle. If enable resetting slider is enabled, the sliders will automatically go back to zero.

Dependencies

This plugin needs the following libraries:

- QT v.5.9.5

Plugin Details

Path	src/aadcDemo/helper/CarController
------	-----------------------------------

Filename	car_controller.adtfplugin
Version	1.0.0

Output Pins

Pin	Description	Media Description
Steering	tSignalValue for the steering controller	percent [-100, 100]
Speed	tSignalValue for the speed controller	percent [-100, 100]
head_light	tBoolSignalValue for the headlight	boolean
turn_signal_left	tBoolSignalValue for the turn left lights	boolean
turn_signal_right	tBoolSignalValue for the turn right lights	boolean
brake_light	tBoolSignalValue for the brake lights	boolean
hazard_light	tBoolSignalValue for the hazard lights	boolean
reverse_light	tBoolSignalValue for the reverse lights	boolean

5.2.2.5 Converter Wheels

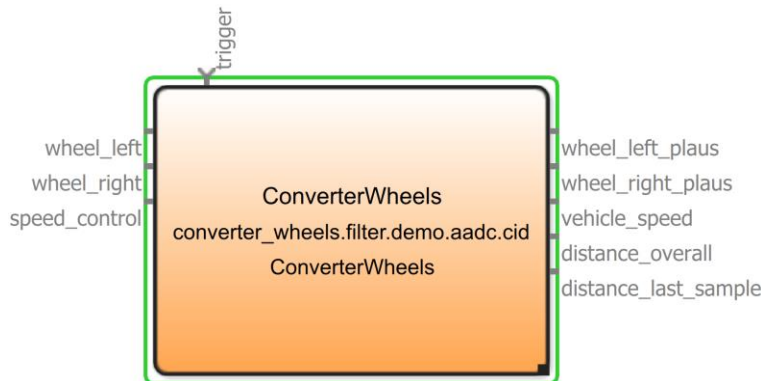


Figure 24 Converter Wheels Plugin

This filter calculates the speed of the vehicle and some distance measurements.

It takes the wheel structs containing the direction and the interrupt counter of both wheels from the Arduino sensor as input values. The wheels have a small disk mounted at the inner side and a Transmissive Encoder Sensor which detects the slots in the rotating disk. The detected slots are transmitted to an interrupt routine at a Arduino which counts the ticks and sends them periodically to ADTF. The speed is calculated by the difference between the increasing tick counter in the incoming media samples and the corresponding time difference between the media samples. Positive speed values indicate driving forward, negative speed values indicate driving backwards. For a correct calculation, the wheel circumference has to be set to the correct value in the properties, the default value is 0.34m. The output samples are triggered to the wheel right struct, so if no samples are received from that sensor no output samples are generated by this filter. If the

property Filtering enabled is set to true a first order filtering is applied to smooth the signals. The first order filter constant can also be set in the properties.

Plugin Details

Path	src/aadcDemo/datafilter/ ConverterWheels
Filename	converter_wheels.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
filter constant of first order	Set the filter constant for first order here	0.3
enable filtering	Enables or disables the low pass filtering of speed result	False
plausibilization via direction indicator enable		True
direction derived from speed controller actuator value		True
deadband for speed controll direction		0.1000000015
wheel circumference [m]	Set the wheel circumference in meter here	0.34

Input Pins

Pin	Description	Media Description
speed_control		tSignalValue
wheel_left		tWheelData
wheel_right		tWheelData

Output Pins

Pin	Description	Media Description
wheel_left_plaus		tWheelData
wheel_right_plaus		tWheelData
vehicle speed		tSignalValue
distance_overall		tSignalValue
distance_last_sample		tSignalValue

5.2.2.6 Driver Module

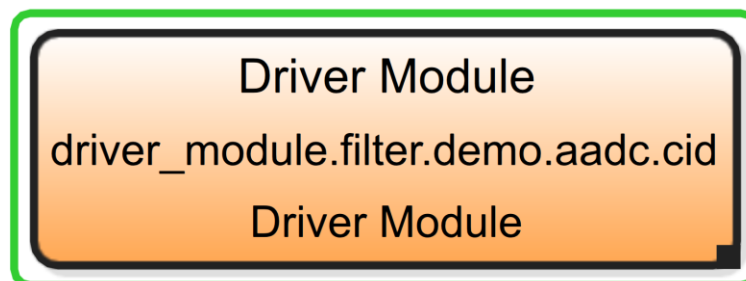


Figure 25 Driver Module Plugin

This filter was developed to explain the interaction with the Jury Module.

It receives the structs from the Jury module containing the actions and maneuvers that the car should perform on the track and shows them in the middle of the gui.

The user can respond the received messages with the five buttons “Ready to Start”, “Running”, “Error” and “Complete”.

The possible i8StateID are:

- stateCar_Error: This is sent if some error occurred on the car.
- stateCar_Ready: If the car is ready to start a maneuver ID this state is sent including the maneuver ID in i16ManeuverEntry.
- stateCar_Running: Sent during running the maneuver contained in i16ManeuverEntry
- stateCar_Complete: Sent if the car finished the whole maneuver list.
- stateCar_Startup: Sent at the initial phase to indicate that car is working properly

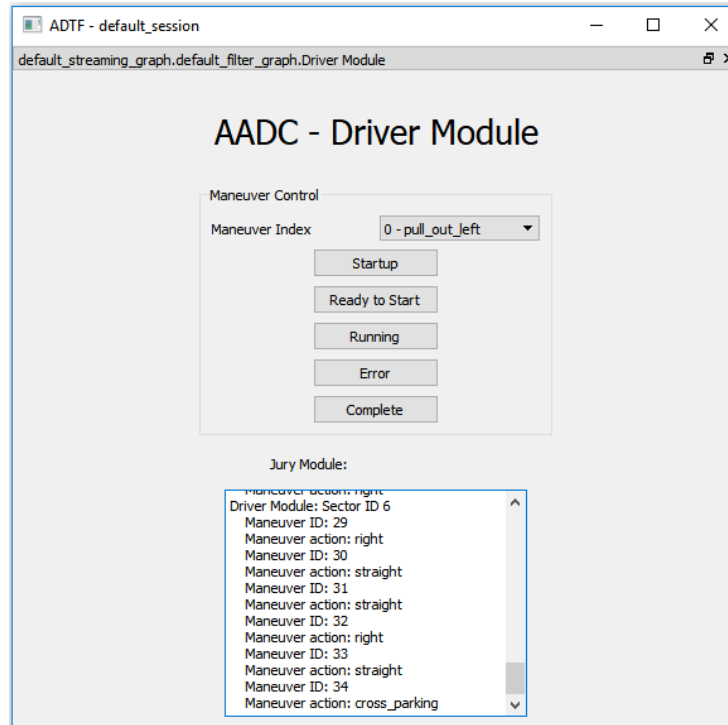


Figure 26 GUI Driver Module

The struct `tDriverStruct` is defined in `aadc_structs.h` in `src`, the used enums are defined in **juryEnums.h** in `src`. The teams must not implement any filter containing a Qt GUI because there is no opportunity to control the car with a GUI in the competition. The only way to interact with the car is through the jury module which is controlled by the jury.

Dependencies

This plugin needs the following libraries:

- QT v.5.9.5

Plugin Details

Path	src/aadcDemo/jury/DriverModule
Filename	Driver_module.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
Debug Output to Console		False

5.2.2.7 Fisheye Undistortion

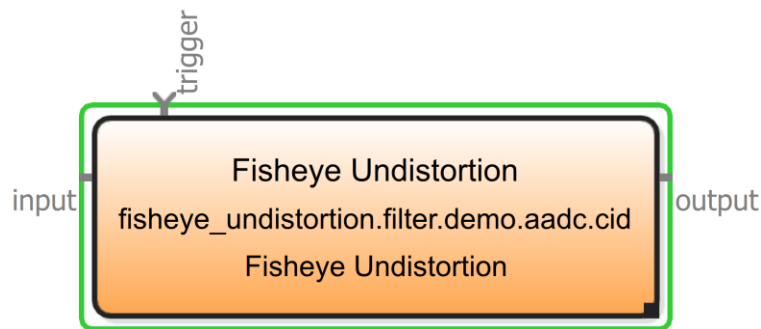


Figure 27 Fisheye Undistortion Plugin

This filter takes an image from a fisheye camera and applies an undistortion to the image.

The filter uses mainly the function `cv::fisheye::undistortImage` from `opencv`. For further information take a look at: https://docs.opencv.org/3.4.1/db/d58/group__calib3d__fisheye.html

Dependencies

This plugin needs the following libraries:

- OpenCV v.3.4.1

Plugin Details

Path	src/aadcDemo/sensor/ FisheyeUndistortion
Filename	fisheye_undistortion.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
calibration file	Set the calibration file here	.
GPU processing	(Only Linux) Use the cuda enabled gpu for image processing.	false
GPU scaling	(Only Linux) If Gpu Processing is enabled, rescale the output image.	1.0

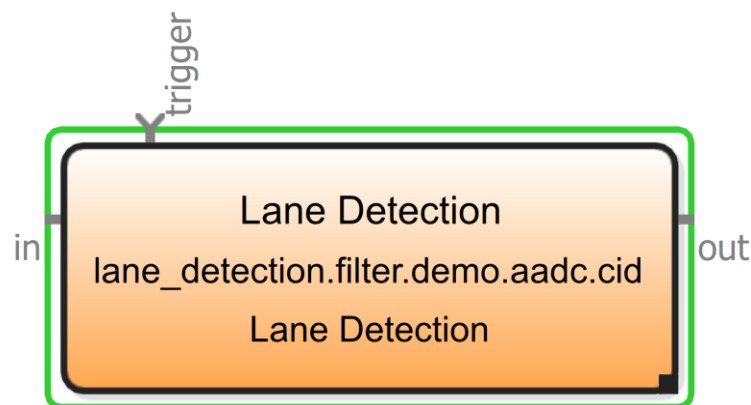
	Recalibrate your camera!	
--	--------------------------	--

Input Pins

Pin	Description	Type
Input	data from camera (distorted)	stream_meta_type_image

Output Pins

Pin	Description	Type
output	undistorted video data	stream_meta_type_image

5.2.2.8 Lane Detection**Figure 28 Lane Detection**

This filter does a very simple detection of lanes in the input image and can be used by the teams as a startup for their own implementation. It does the following subsequent steps:

- binarize input image with threshold set in the properties "Threshold for image binarization"
- calculate the horizontal lines where to search for the Lanes. This is defined by the ROI and the number " detectionLines " set in the filter properties
- iterate through the detection lines and find transitions with high contrasts. For each detection line we iterate from left to right and search for lines within the maximum and minimum width defined in " minLineWidth " and " maxLineWidth "
- all the found linepoints are added to one vector
- suggested: do a classification which point is left line, middle line, right line
- suggested: calculate a clothoid for each line
- suggested: calculate a clothoid for the car

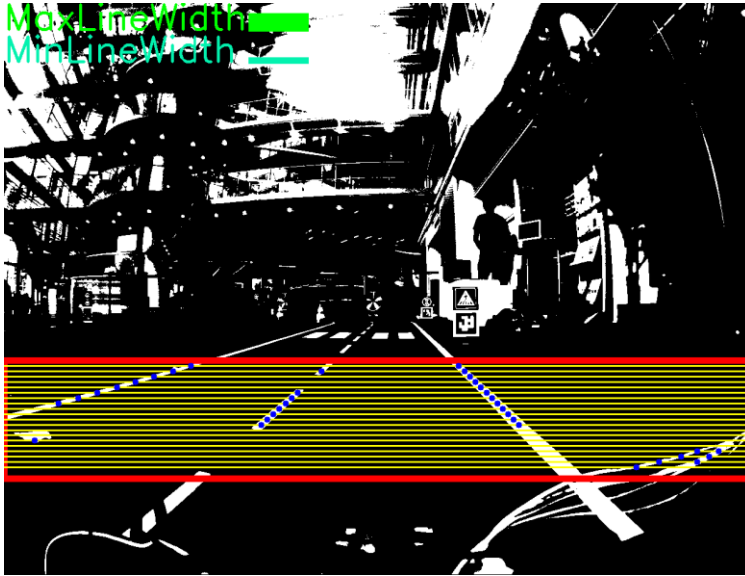


Figure 29 Lane detection filter output showing the line points found on the detection lines within the defined region of interest and with the predefined boundaries for the line width.

Dependencies

This plugin needs the following libraries:

- OpenCV v.3.4.1

Plugin Details

Path	src/algorithms/LaneDetection
Filename	lane_detection.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
ROIOffsetX [Pixel]	X Offset for Region of Interest Rectangular	0
ROIOffsetY [Pixel]	Y Offset for Region of Interest Rectangular	500
ROIWidth [Pixel]	Width of the Region of Interest Rectangular	1280
ROIHeight [Pixel]	Height of the Region of Interest Rectangular	200
detectionLines	number of detection lines searched in ROI	10
minLineWidth [Pixel]	Minimum Line Width in Pixel	10
maxLineWidth [Pixel]	Maximum Line Width in Pixel	30
minLineContrast	Minimum line contrast in gray Values	50
thresholdImageBinarization	Threshold for image binarization	180

Input Pins

Pin	Description	Type
in	undistorted video data	stream_meta_type_image

Output Pins

Pin	Description	Type
Out	debug output showing the detected line points on the image	stream_meta_type_image

5.2.2.9 Marker Detector

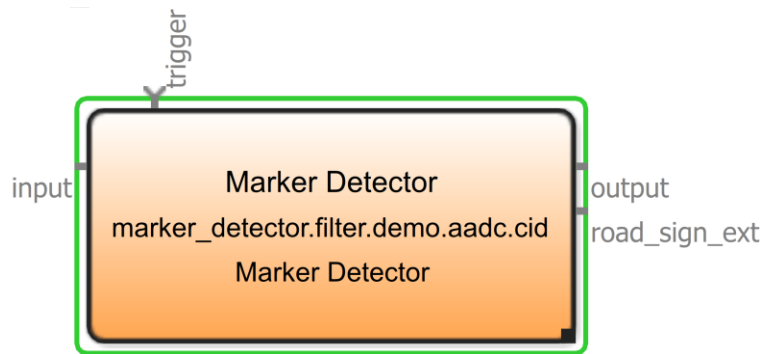



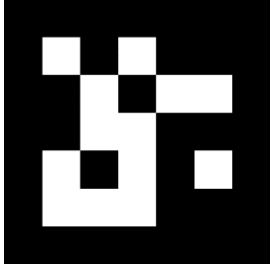


Figure 30 Marker Detector Plugin


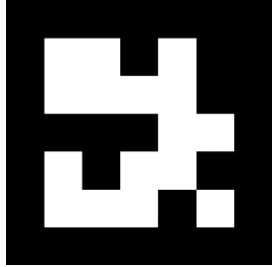

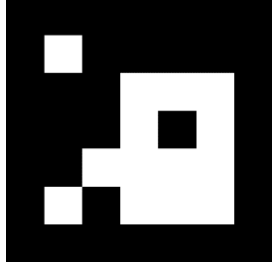

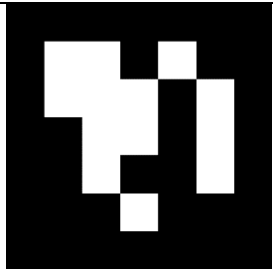

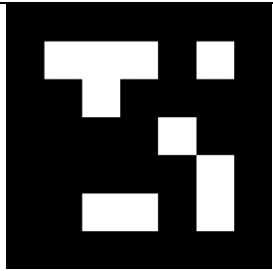

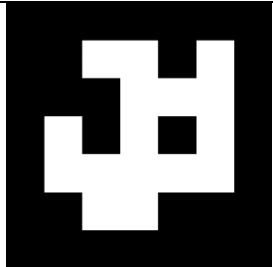

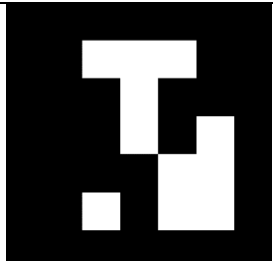
This filter searches for Aruco markers in the given frame on the input video pin.


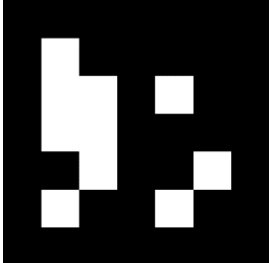

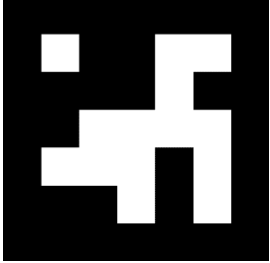
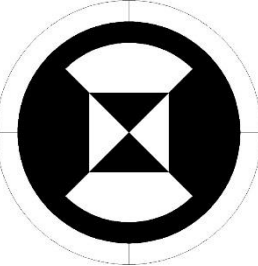
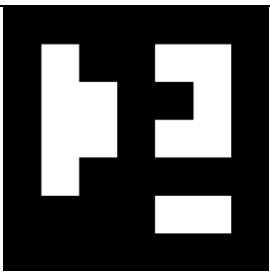

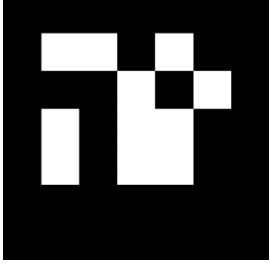

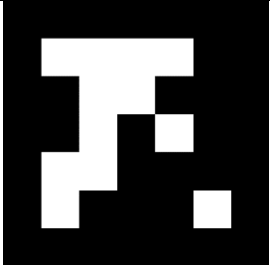

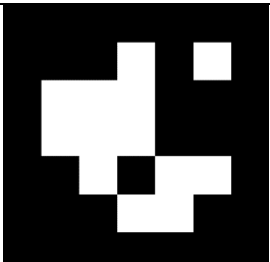
It uses mainly the Aruco lib to find the markers and get the marker Id and their additional parameters and mark them in the video frame. If one or more markers are detected in the frame samples on the pins `road_sign_ext` is generated containing the parameters of the sign. The samples of the pin `road_sign_ext` include the marker identifier, the image size of the marker and the rotation and translation vector.

For the calculation of the rotation and translation vector the intrinsic parameters of the camera has to be known. These parameters can be obtained with the camera calibration filter and the calibration file generated by that filter loaded with in the property Calibration File for the used Camera.

The following table shows an detectable markers and their corresponding traffic signs.

Name	ID	Traffic Sign	Code
UNMARKEDINTERSECTION	0		
STOPANDGIVEWAY	1		

PARKINGAREA	2		
HAVEWAY	3		
AHEADONLY	4		
GIVEWAY	5		
PEDESTRIANCROSSING	6		
ROUNDAABOUT	7		

NOOVERTAKING	8		
NOENTRYVEHICULARTRAFFIC	9		
TESTCOURSEA9	10		
ONEWAYSTREET	11		
ROADWORKS	12		
KMH50	13		

KMH100	14		
--------	----	--	---

Dependencies

This plugin needs the following libraries:

- OpenCV v.3.4.1

Plugin Details

Path	src/aadcDemo/marker/MarkerDetection
Filename	marker_pos.adtfplugin
Version	1.0.0

Plugin Properties

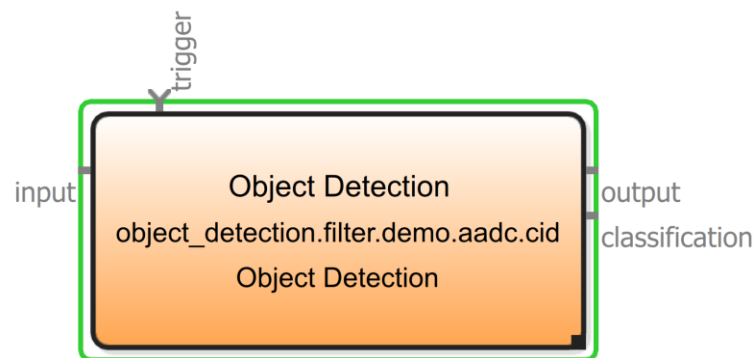
Property	Description	Default
Calibration File	Here you have to set the file with calibration parameters of the used camera	.
Detector Paramater File	Here you have to set the file with the parameters with the detector params	.
Marker Size [m]	Size (length of one side) of markers	0.117f
Marker Roi::x	The x offset of the Region of Interest wherein aruco markers are searched	0.0f
Marker Roi::y	The x offset of the Region of Interest wherein aruco markers are searched	0.0f
Marker Roi::height	The height of the Region of Interest wherein aruco markers are searched	10.0f
Marker Roi::width	The width of the Region of Interest wherein aruco markers are searched	10.0f
Show Roi	Display the Region of Interest on the output image. Decreases performance if active.	tFalse

Input Pins

Pin	Description	Type
Input	video input from camera	stream_meta_type_image

Output Pins

Pin	Description	Type
Output	Debug output showing identified markers on the image with the rotation and translation vector	stream_meta_type_image
road_sign_ext	roadsigns with extended information	tRoadSignExt

5.2.2.10 Object Detection**Figure 31 Object Detection Filter**

This filter uses a pre-trained Caffe neuronal network model to classify objects appearing on image data of the input pin according to the classes specified in the classNames property file.

Dependencies

This plugin needs the following libraries:

- OpenCV v.3.4.1

Plugin Details

Path	src/aadcDemo/algorithms/ObjectDetecion
Filename	object_detection.adtfplugin
Version	1.0.0

Plugin Properties

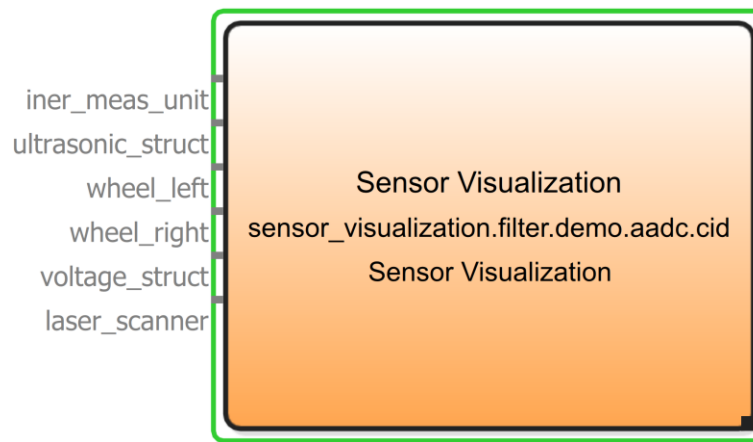
Property	Description	Default
classNames	List which assigns each class label a human understandable class description	synset_words.txt
Model	The learned Caffe model	bvlc_googlenet.caffemodel
Proto	text description of the network architecture	bvlc_googlenet.prototxt
opencl	Use opencl for GPU support	tTrue

Input Pins

Pin	Description	Type
Input	Video data input	stream_meta_type_image

Output Pins

Pin	Description	Media Description
output	Debug output showing detected objects	stream_meta_type_image
classification	Classification results in DDL	tClassification

5.2.2.11 Sensor Visualization**Figure 32 Sensor Visualization Plugin**

This filter opens a QT window, which visualizes most of the sensor data. Thereon several tables are shown including the data from the ultrasonic sensors, laser scanner, from the wheel speed sensors, from the voltage sensors and from the inertial measurement sensor (IMU). On the left side of the windows is also a graphical scene illustrating the values from the ultrasonic sensors and the laser scanner data.

Dependencies

This plugin needs the following libraries:

- QT v.5.9.5

Plugin Details

Path	src/aadcDemo/helper/SensorVisualization
Filename	sensor_visualization.adtfplugin
Version	1.0.0

Input Pins

Pin	Description	Media Description
iner_meas_unit		tInerMeasUnitData
ultrasonic_struct		tUltrasonicStruct
wheel_left		tWheelData

wheel_right		tWheelData
voltage_struct		tVoltageStruct
laser_scanner		tLaserScannerData

5.2.2.12 Wheel Speed Controller

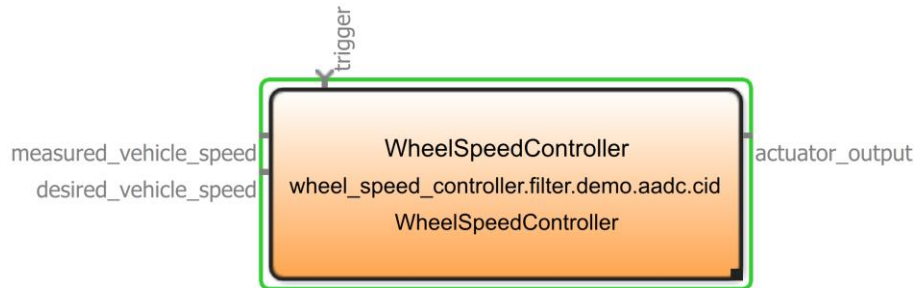


Figure 33 Wheel Speed Controller Plugin

This filter implements a controller to set the vehicle speed of the vehicle with a P/PI/PID or PT1 algorithm. The input pin `measured_vehicle_speed` has to be connected to the output pin of the Converter Wheels Filter and the desired wheel speed has to be set to the input pin `desired_vehicle_speed`. The controller parameters have to be adapted to each individual car. The default values for the PT1 controller are good start for the controller but maybe have to be adapted to the individual team car. There are different methods to get the correct parameter, please refer to other literature. The output pin `actuator_output` has to be connected to the Arduino Actuator streaming sink speed pin.

Plugin Details

Path	src/aadcDemo/algorithms/ WheelSpeedController
Filename	wheel_speed_controller.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
controller type	P controller: 1 PI controller: 2 PID controller: 3 PT1 controller: 4	2
show debug output	If true debug infos are plotted to registry	tFalse
differential factor for PID Controller	The differential factor Kd for the PID Controller	0.01
integral factor for PID Controller	The integral factor Ki for the PID Controller	0.85
proportional factor for PID Controller	The proportional factor Kp for the PID Controller	10
the maximum output value for the controller [%]	The maximum allowed output for the wheel speed controller	20
the minimum output value for the controller [%]	The minimum allowed output for the wheel speed controller	-20
sampletime for the pid	The sample interval in ms used	0.025

controller [ms]	by the PID controller	
set point is multiplied with this factor	Correction factor for input set point	1.15
gain factor for PT1 controller	Gain for PT1 Controller	14
input factor for PT1	The factor to normalize the output value	1
time constant for pt1 controller	Time Constant for PT1 Controller	1

Input Pins

Pin	Description	Media Description
desired_vehicle_speed		tSignalValue
measured_vehicle_speed		tSignalValue

Output Pins

Pin	Description	Media Description
actuator_output		tSignalValue

5.3 AADC User

5.3.1 OpenCV Template Filter

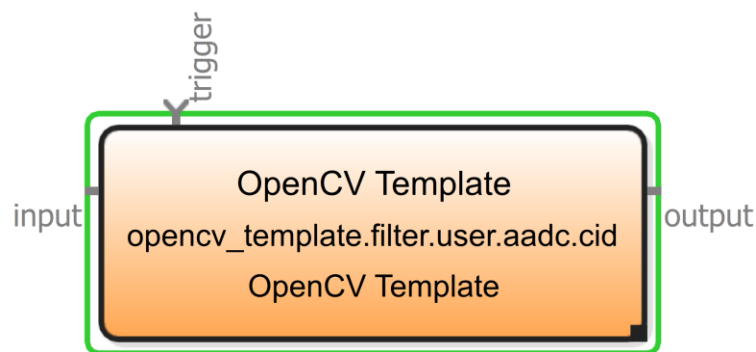


Figure 34 OpenCV Template Filter

This is a small OpenCV template which can be used by the AADC teams for their own filter implementations for image processing. It applies a simple OpenCV edge detection filter to the input video.

Dependencies

This plugin needs the following libraries:

- OpenCV v.3.4.1

Plugin Details

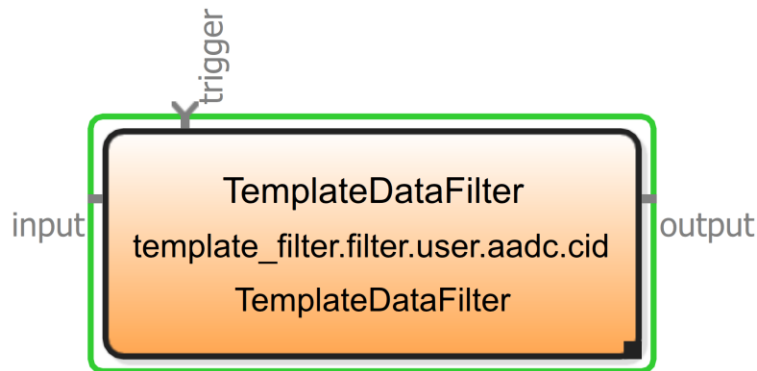
Path	src/aadcUser/OpenCVTemplate
Filename	opencv_template.adtfplugin
Version	1.0.0

Input Pins

Pin	Description	Type
Input	Video Pin for data from camera	stream_meta_type_image

Output Pins

Pin	Description	Type
output	Output pin for processed video data	stream_meta_type_image

5.3.2 User Template Plugin**Figure 35 User Template Filter**

This is a small template which can be used by the AADC teams for their own filter implementations.

Plugin Details

Path	src/aadcUser/TemplateFilter
Filename	template_filter.adtfplugin
Version	1.0.0

Plugin Properties

Property	Description	Default
----------	-------------	---------

Input Pins

Pin	Description	MediaDescription
Input	An example input pin	tTemplateData

Output Pins

Pin	Description	MediaDescription
output	An example output pin for the processed data	tTemplateData

5.4 How to program your first plugin

Students team are advised to work in the delivered ADTF Source package. When doing this the teams do not have to set up their own CMake project or worry about the dependencies.

1. Go to the folder src\aadcUser
2. Copy one of the template filters or any other example from src\aadcDemo
3. Rename folder

4. Add renamed folder to end of `src\aadcUser\CMakelists.txt`:
 `add_subdirectory(AADC_TeamFilter)`
5. Rename files in new folder
6. Rename filter ID and class name in `.cpp` and `.h`
7. Build user folder (run `build_user.sh`)
8. Start user project again.

6 ADTF Projects and Sessions

The ADTF Configuration Editor must be started using the script “start_adtf_ce_editor.sh”, which sets all the necessary environment variables.

6.1 Live Visualization Project

The template project “LiveVisualization” is located in the “config” folder and contains 6 template sessions. This project can be used in order to check if all sensors and actuator are working properly and as a starting point for developing own functionalities. However be aware that all debug related services (sample stream trace view, log view, signal views, ...) are activated in the system, this will decrease performance.

ATTENTION: When you load a session in the ADTF 3 Configuration Editor and switch to the Streaming Graph Editor / Filter Graph Editor the session specific Streaming Graph / Filter Graph is not automatically displayed but has to be selected in the tabs.

Following, all available sessions are briefly described.

6.1.1 default_session

This session contains the full available ADTF representation of the car with all sensors, their visualization and actuators. The following streaming graph and filter graph are set for this session:

6.1.1.1 default_streaming_graph

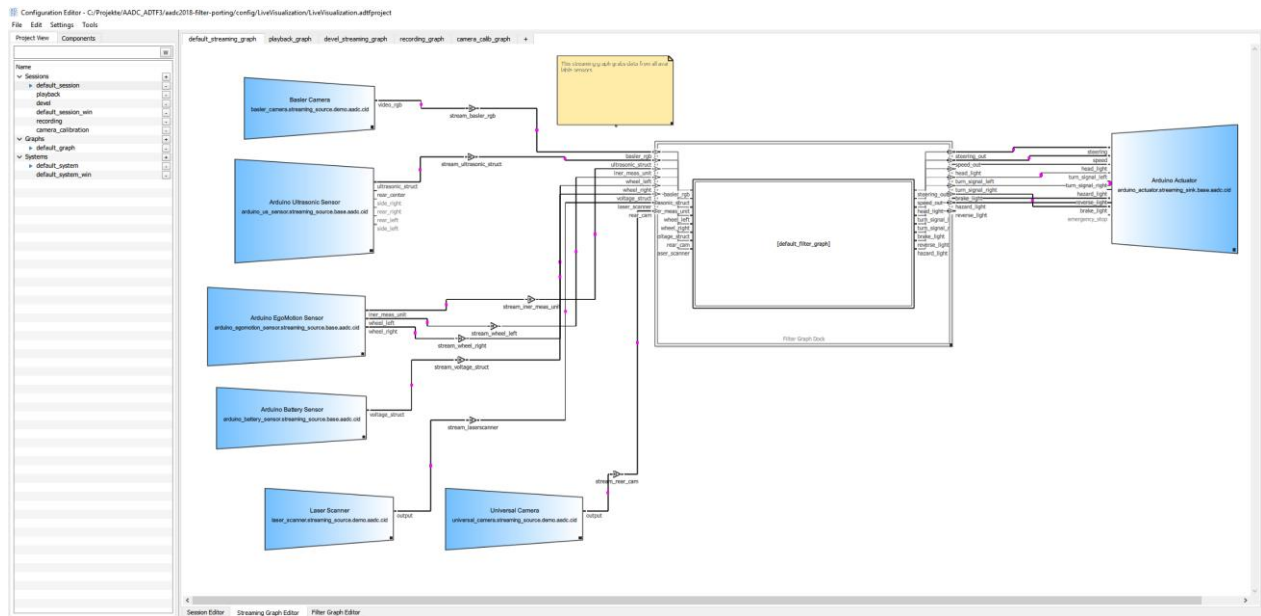


Figure 36 default_streaming_graph

6.1.1.2 default_filter_graph

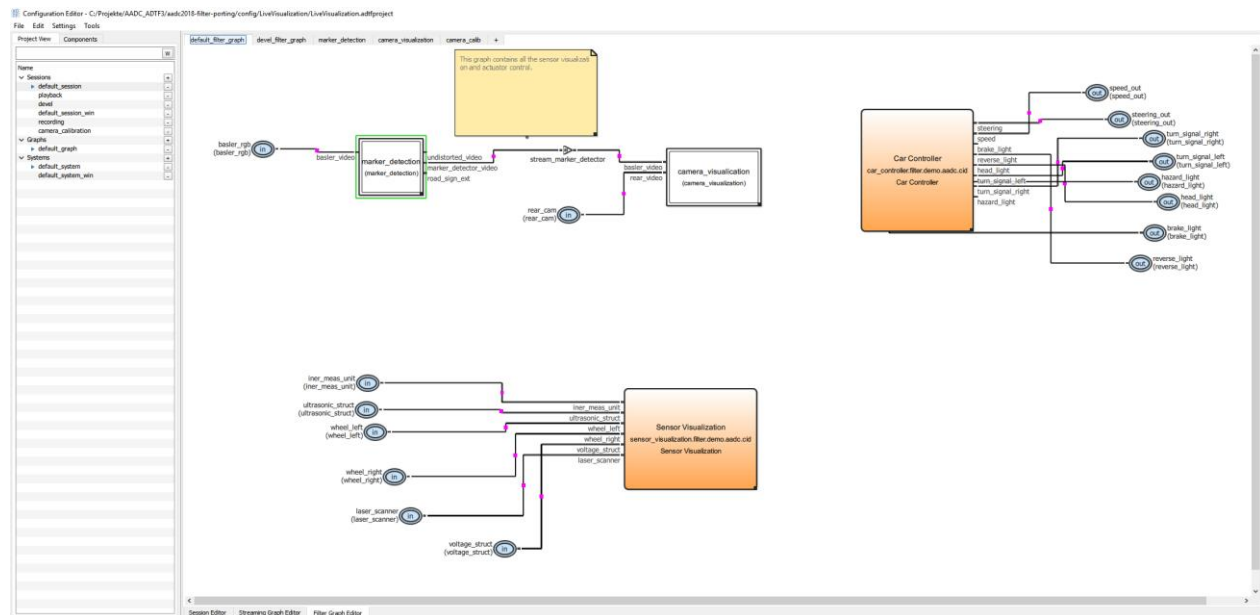


Figure 37 default_filter_graph

When you start the session, the following interface appears:

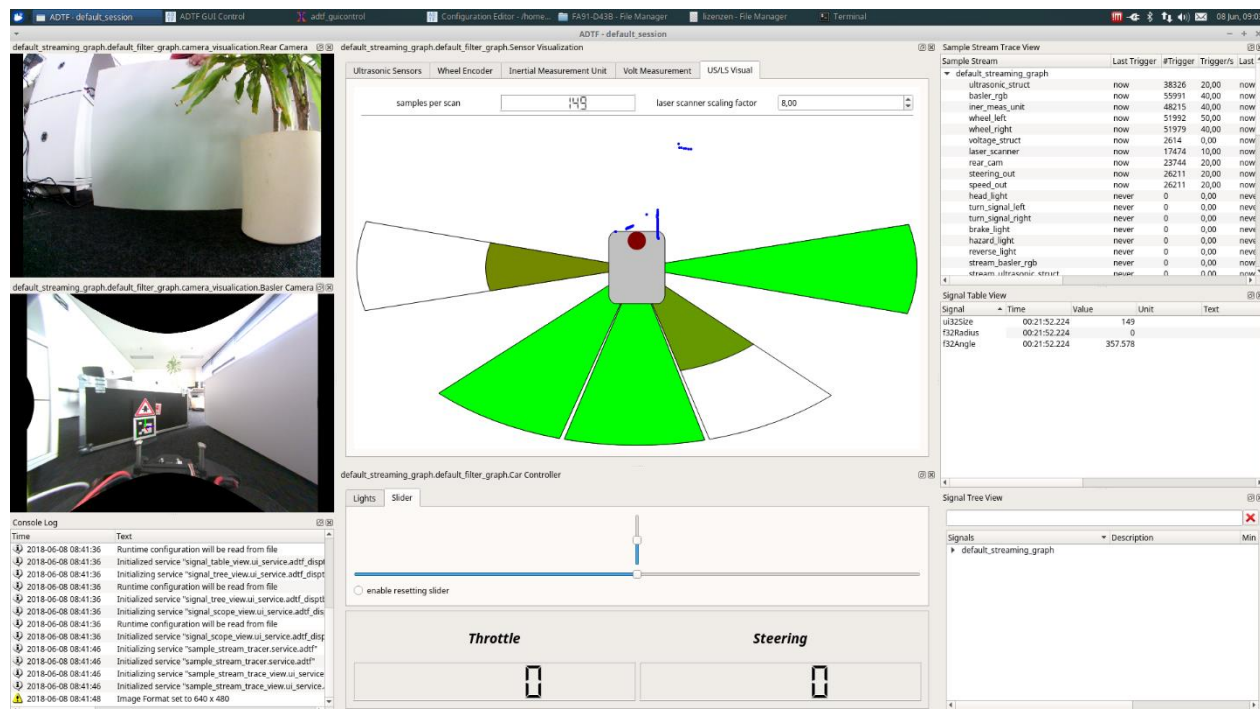


Figure 38 Running session view

6.1.2 recording

This session can be used to play record sensor data and save the record in an .adtfdat file.

6.1.2.1 recording_graph

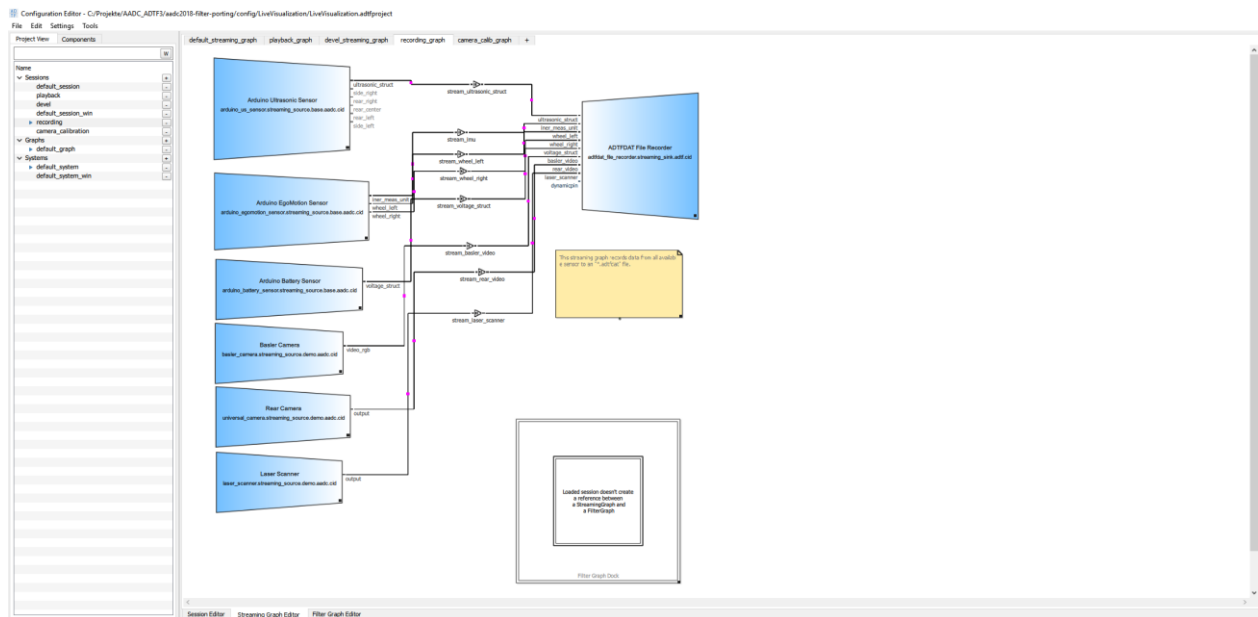


Figure 39 recording_graph filter graph

6.1.3 playback

This session can be used to play recorded .adtfdat files and display the sensor values in the default sensor visualization.

6.1.3.1 playback_graph

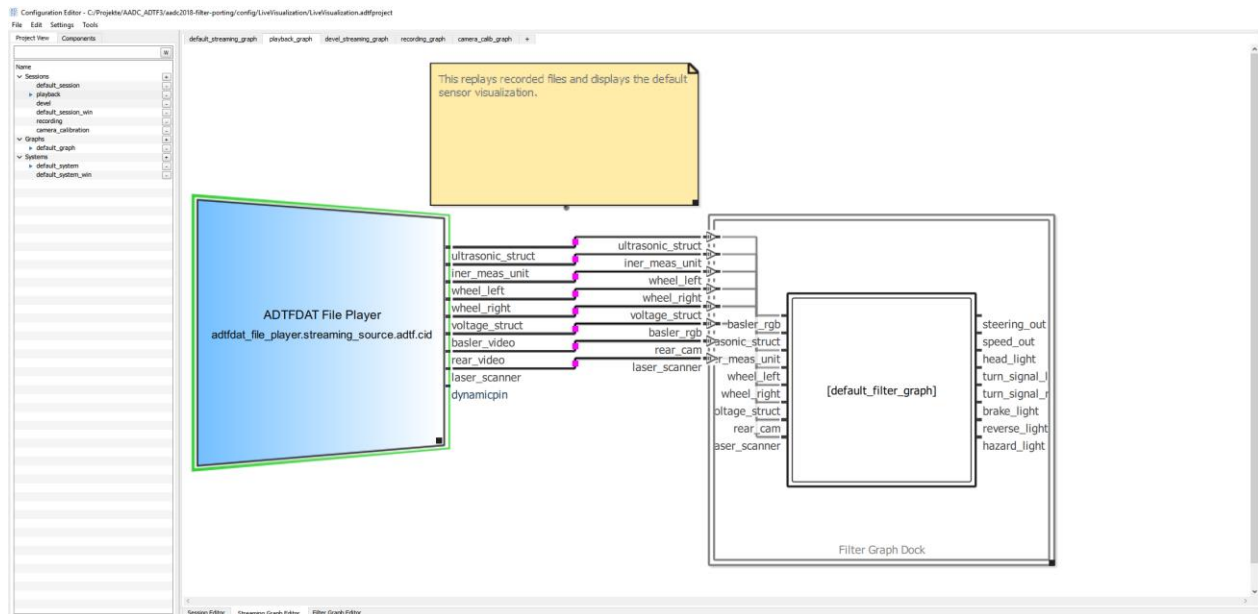


Figure 40 playback_graph streaming graph

6.1.3.2 default_filter_graph

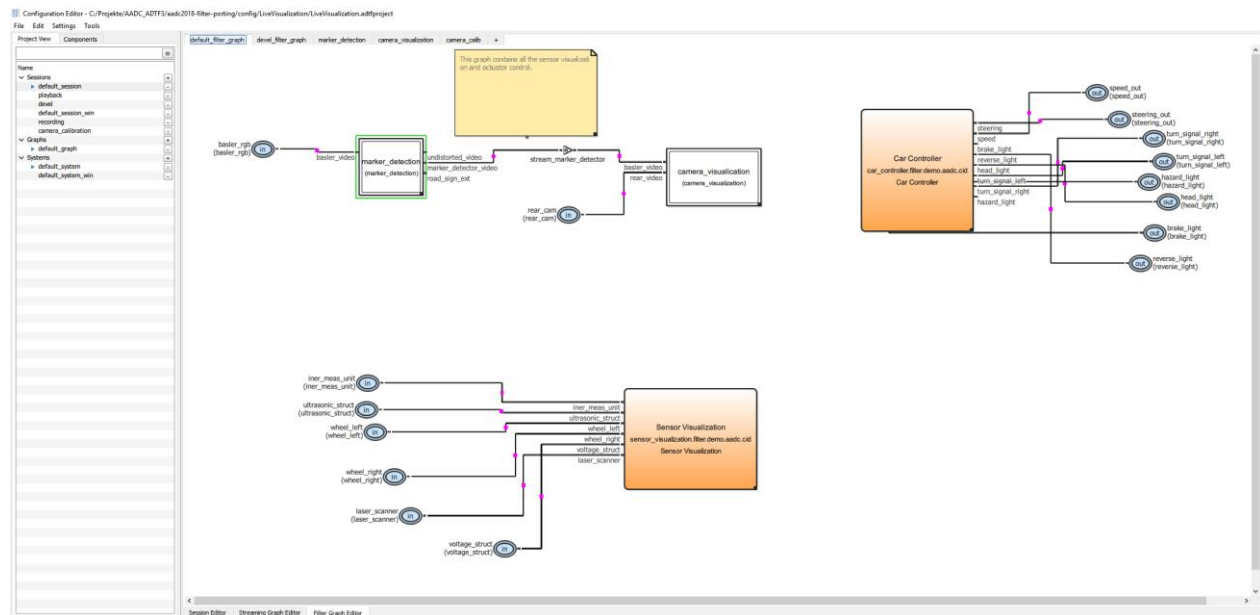


Figure 41 default_filter_graph

6.1.4 devel

This session can be used as a template for own streaming graph / filter graph development.

6.1.4.1 devel_streaming_graph

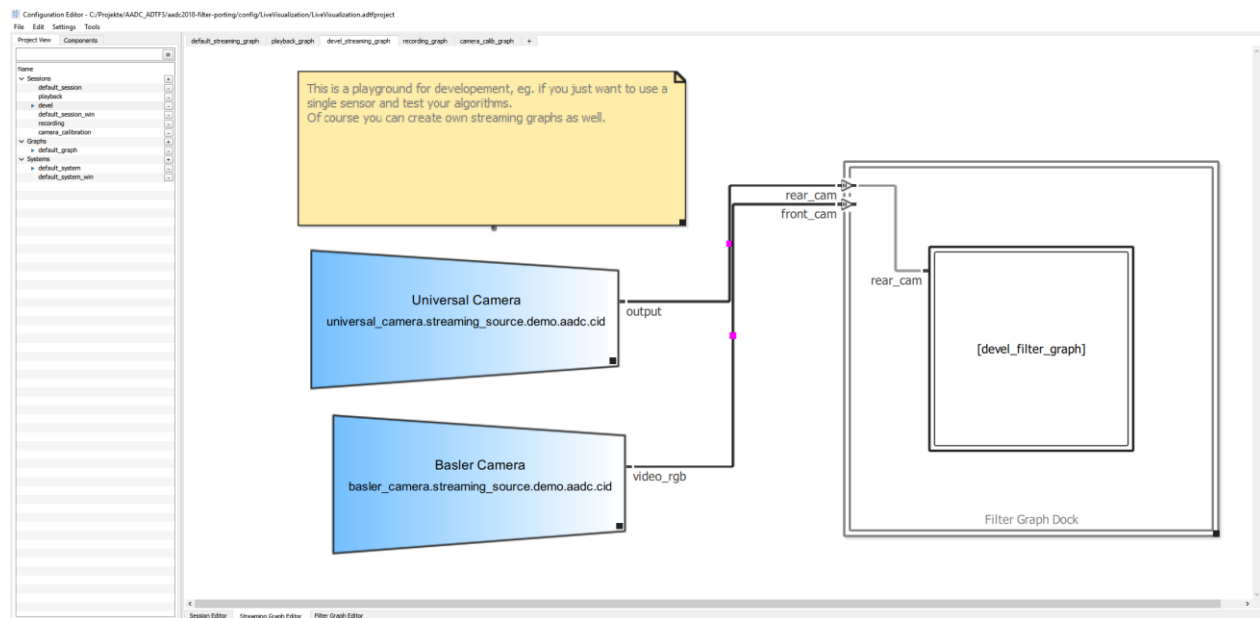


Figure 42 devel_streaming_graph

6.1.4.2 devel_filter_graph

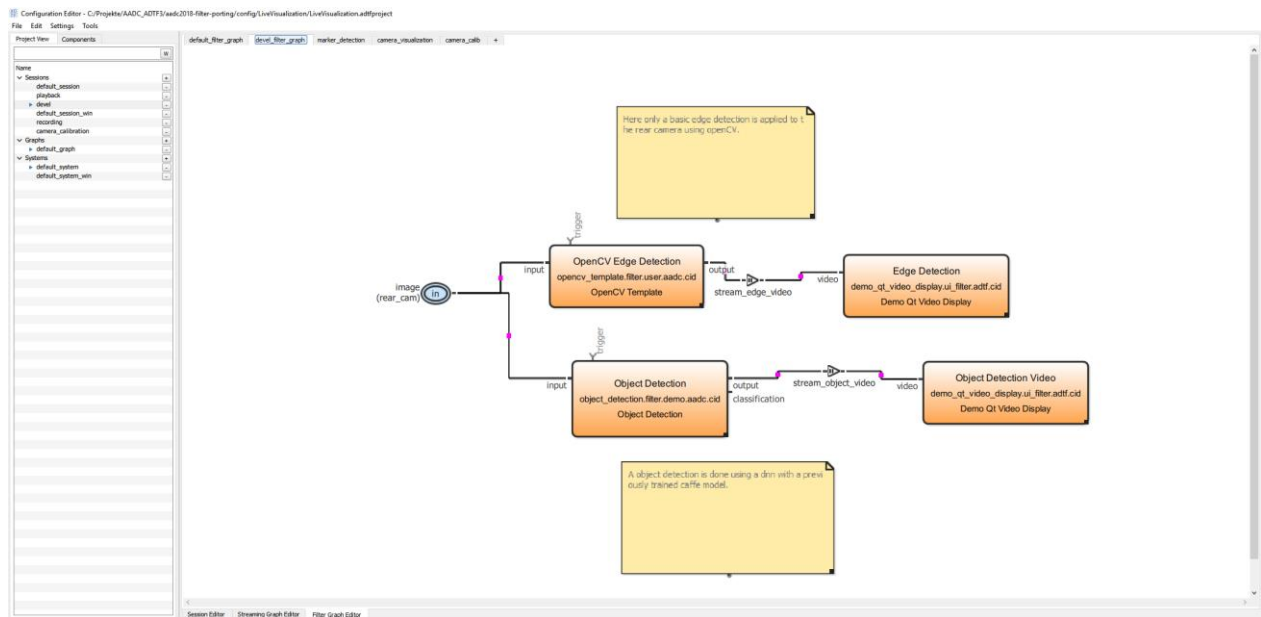


Figure 43 devel_filter_graph

When you start the session, the following interface appears:

Console Log

Time	Text	Source
2018-06-08 09:05:09	Initializing service "log_view.ui_service.adtf"	runtime.cpp(777)
2018-06-08 09:05:09	Initializing service "adtf_file_support.service.adtf"	runtime.cpp(746)
2018-06-08 09:05:09	Initializing service "adtf_file_support.service.adtf"	runtime.cpp(777)
2018-06-08 09:05:09	Initializing service "playback.service.adtf"	runtime.cpp(746)
2018-06-08 09:05:09	Initializing service "playback.service.adtf"	runtime.cpp(777)
2018-06-08 09:05:17	Initializing service "signal_registry.service.adtf"	runtime.cpp(746)
2018-06-08 09:05:17	Initializing service "signal_registry.service.adtf"	runtime.cpp(777)
2018-06-08 09:05:17	Initializing service "media_description.service.adtf"	runtime.cpp(746)
2018-06-08 09:05:17	Initializing service "media_description.service.adtf"	runtime.cpp(777)
2018-06-08 09:05:17	Initializing service "signal_table_view.ui_service.adtf_disptb"	runtime.cpp(746)
2018-06-08 09:05:17	Runtime configuration will be read from file	runtimeconfig.h(156)
2018-06-08 09:05:17	Initializing service "signal_table_view.ui_service.adtf_disptb"	runtime.cpp(777)
2018-06-08 09:05:17	Initializing service "signal_tree_view.ui_service.adtf_disptb"	runtime.cpp(746)
2018-06-08 09:05:17	Runtime configuration will be read from file	runtimeconfig.h(156)
2018-06-08 09:05:17	Initializing service "signal_tree_view.ui_service.adtf_disptb"	runtime.cpp(777)
2018-06-08 09:05:17	Initializing service "signal_scope_view.ui_service.adtf_disptb"	runtime.cpp(746)
2018-06-08 09:05:17	Runtime configuration will be read from file	runtimeconfig.h(156)
2018-06-08 09:05:17	Initializing service "signal_scope_view.ui_service.adtf_disptb"	runtime.cpp(777)
2018-06-08 09:05:17	Loaded caffe model successfully from /home/luad/AADC/config/Live	ObjectDetection.cpp(156)
2018-06-08 09:05:17	Initializing service "sample_stream_tracer.service.adtf"	runtime.cpp(746)
2018-06-08 09:05:17	Initializing service "sample_stream_tracer.service.adtf"	runtime.cpp(777)
2018-06-08 09:05:17	Initializing service "sample_stream_trace_view.ui_service.adtf"	runtime.cpp(746)
2018-06-08 09:05:17	Initializing service "sample_stream_trace_view.ui_service.adtf"	runtime.cpp(777)
2018-06-08 09:05:17	Image Format set to 1280 x 960	UniversalCamera.cpp(87)

Sample Stream Trace View

Sample Stream	Last Trigger	#Trigger	Trigger/s	Last Sample	#Samples	Sample/s	MB/s	Source
devel_streaming_graph	now	515	0.00	now	516	10.00	35...	[pin] devel_streaming_graph.Universal Camera output
rear_cam	now	794	10.00	now	794	10.00	35...	[pin] devel_streaming_graph.Buster Camera video: rgb
devel_filter_graph	now	515	0.00	now	516	10.00	35...	[sample stream] devel_streaming_graph.rear_cam
image	now	1030	0.00	now	515	0.00	0.00	[pin] devel_streaming_graph.devel_filter_graph.Object Detection
stream_object_video	now	1032	20.00	now	516	10.00	11...	[pin] devel_streaming_graph.devel_filter_graph.OpenCV Edge

Figure 44 Running session view

6.1.5 camera_calibration

Use this session to calibrate the fisheye camera.

6.1.5.1 camera_calib_graph

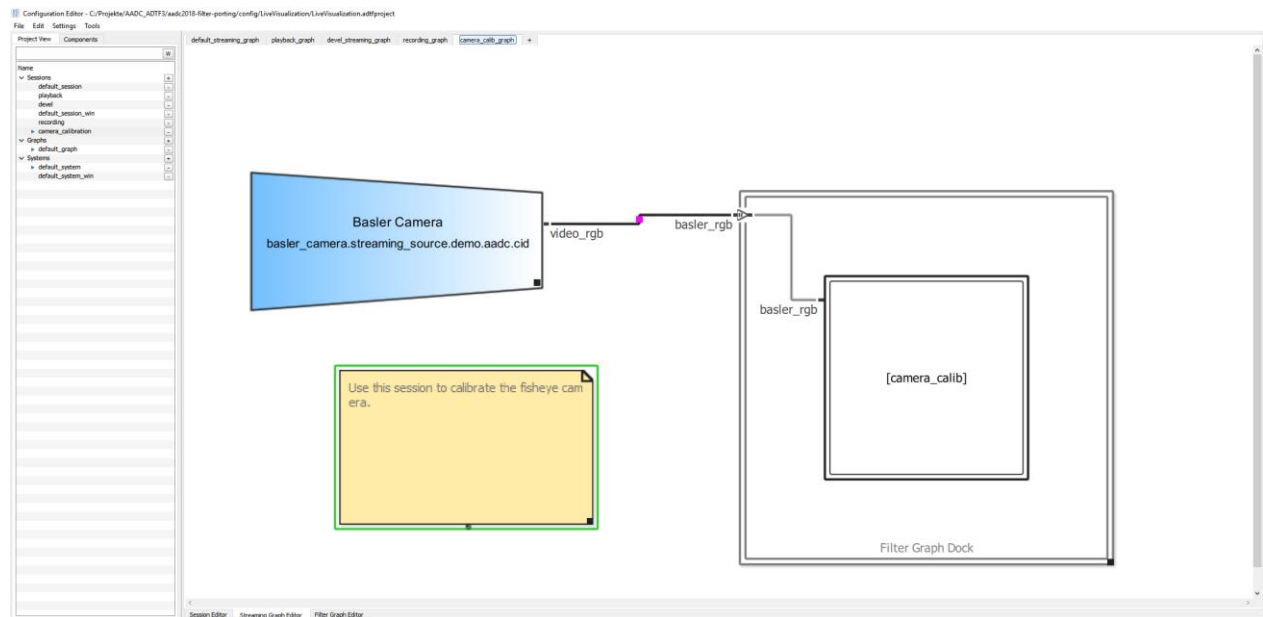


Figure 45 camera_calib_graph streaming graph

6.1.5.2 camera_calib

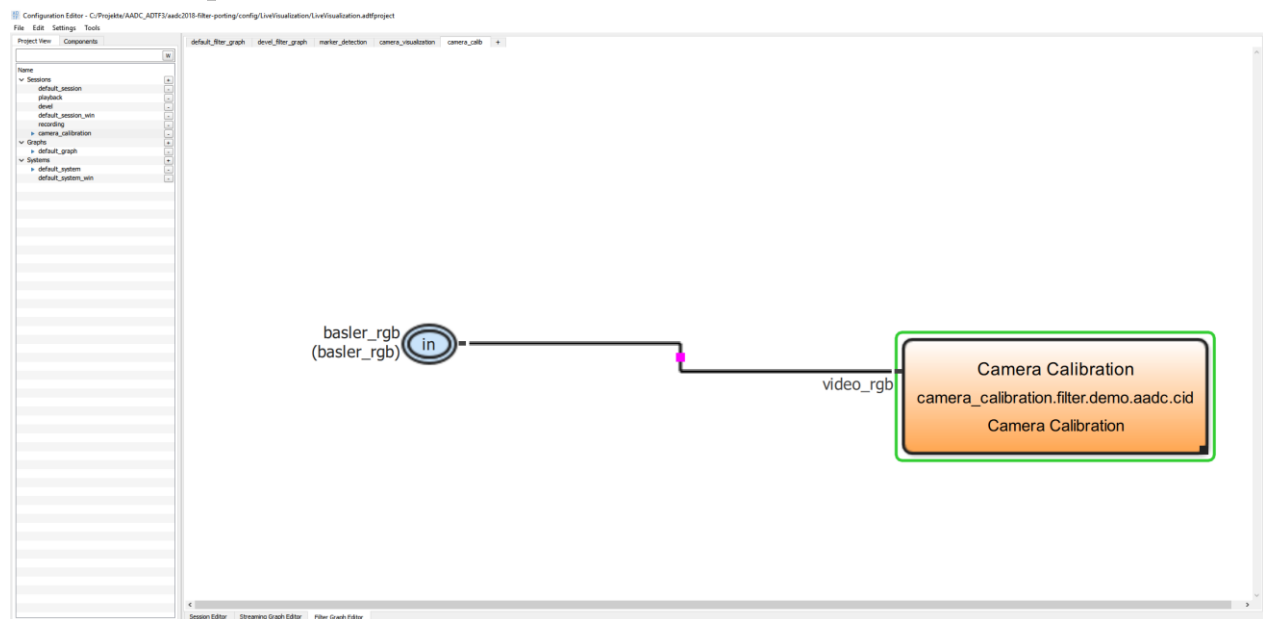


Figure 46 camera_calib filter graph

When you start the session, the following user interface appears:

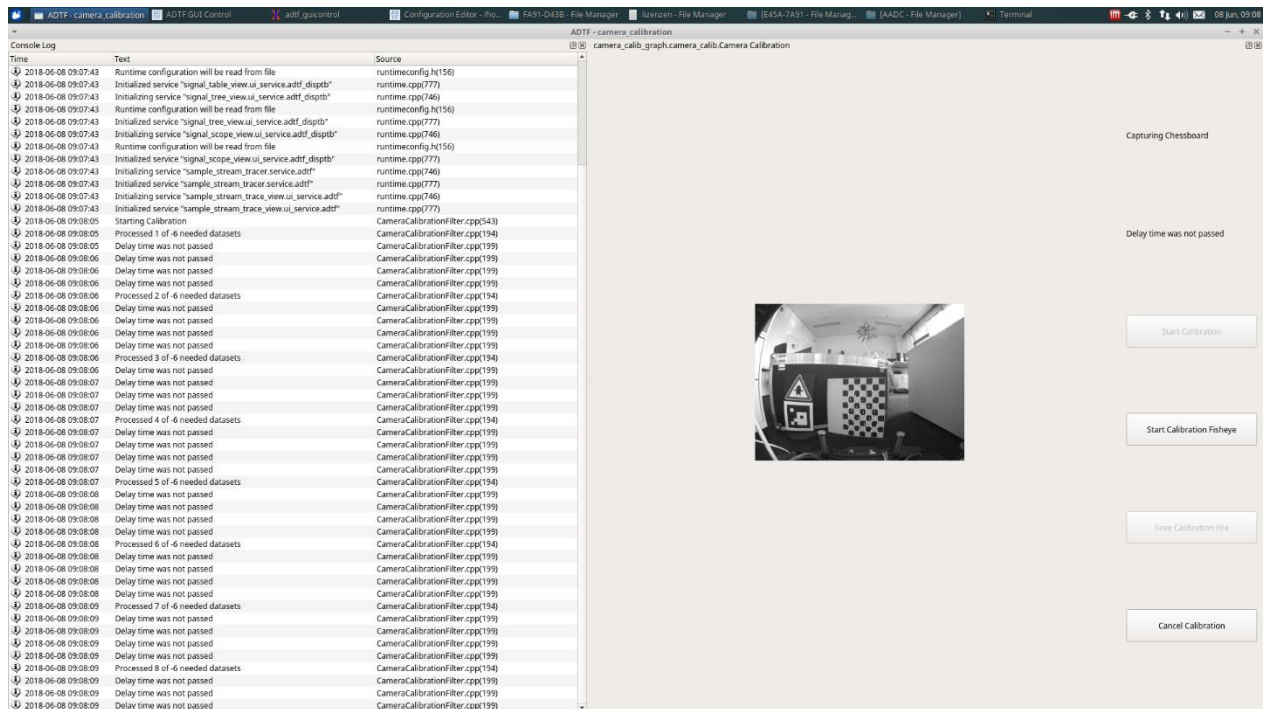


Figure 47 Running session view

6.1.6 default_session_win

When running ADTF3 on a Microsoft Windows machine this is the default_session equivalent to be used.

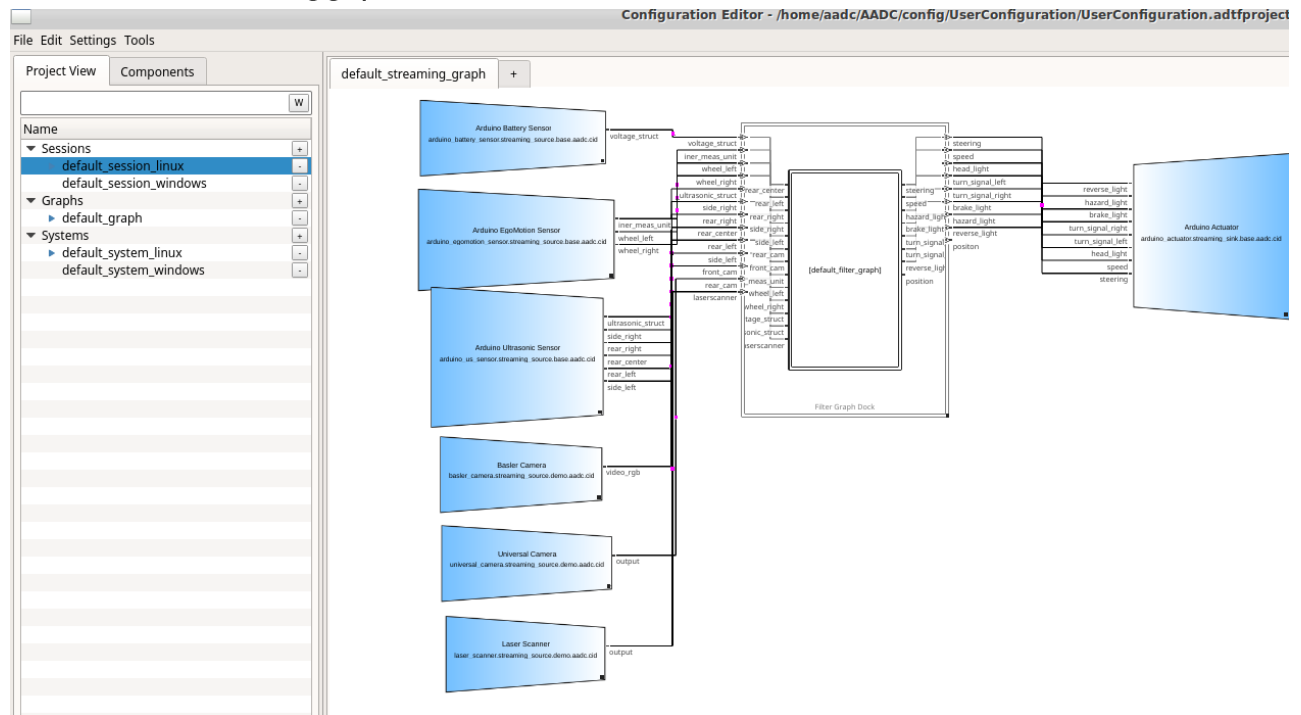
6.2 User Configuration Project

The template project “UserProject” is located in the “config” folder and contains two template sessions. This project can be used as a template project for a competition configuration, for instance without all the debug visualization services/filters activated and thus optimized for performance. However the map visualization filter should not be part of a competition project as well as the Driver Module which has to be replaced by a custom implementation preferable without any GUI.

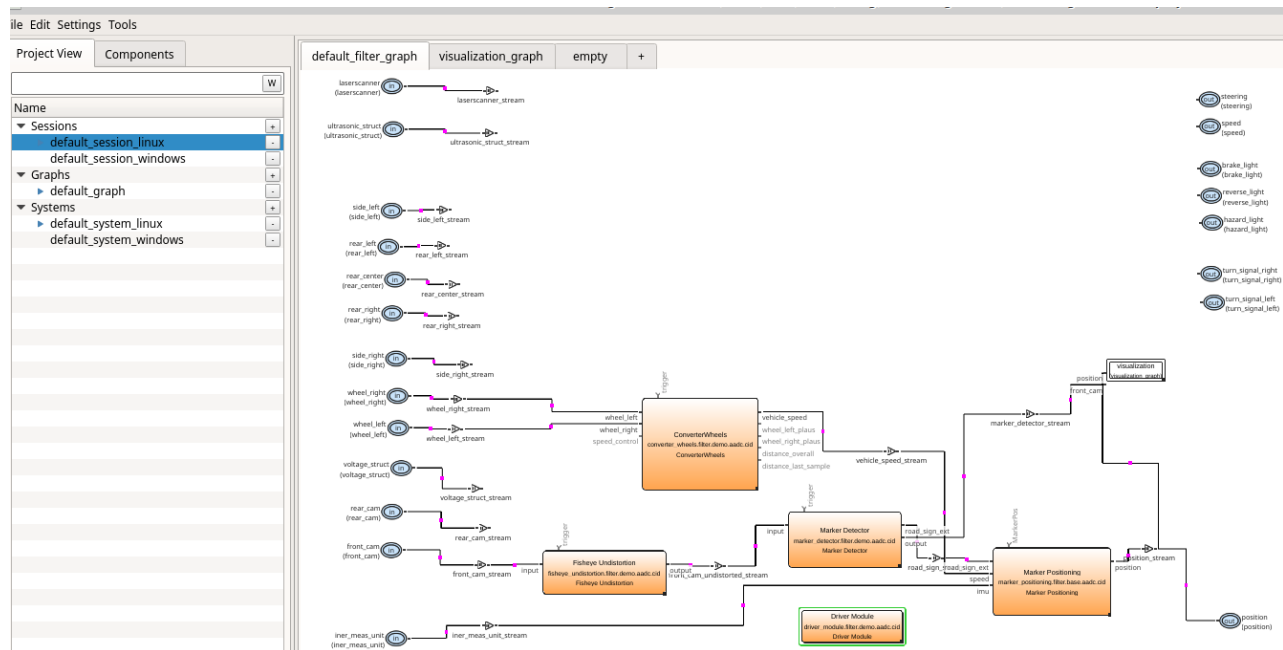
6.2.1 default_session

There are only two sessions in this project both with the same streaming and filter graph only the system file either for linux or for windows with its corresponding paths.

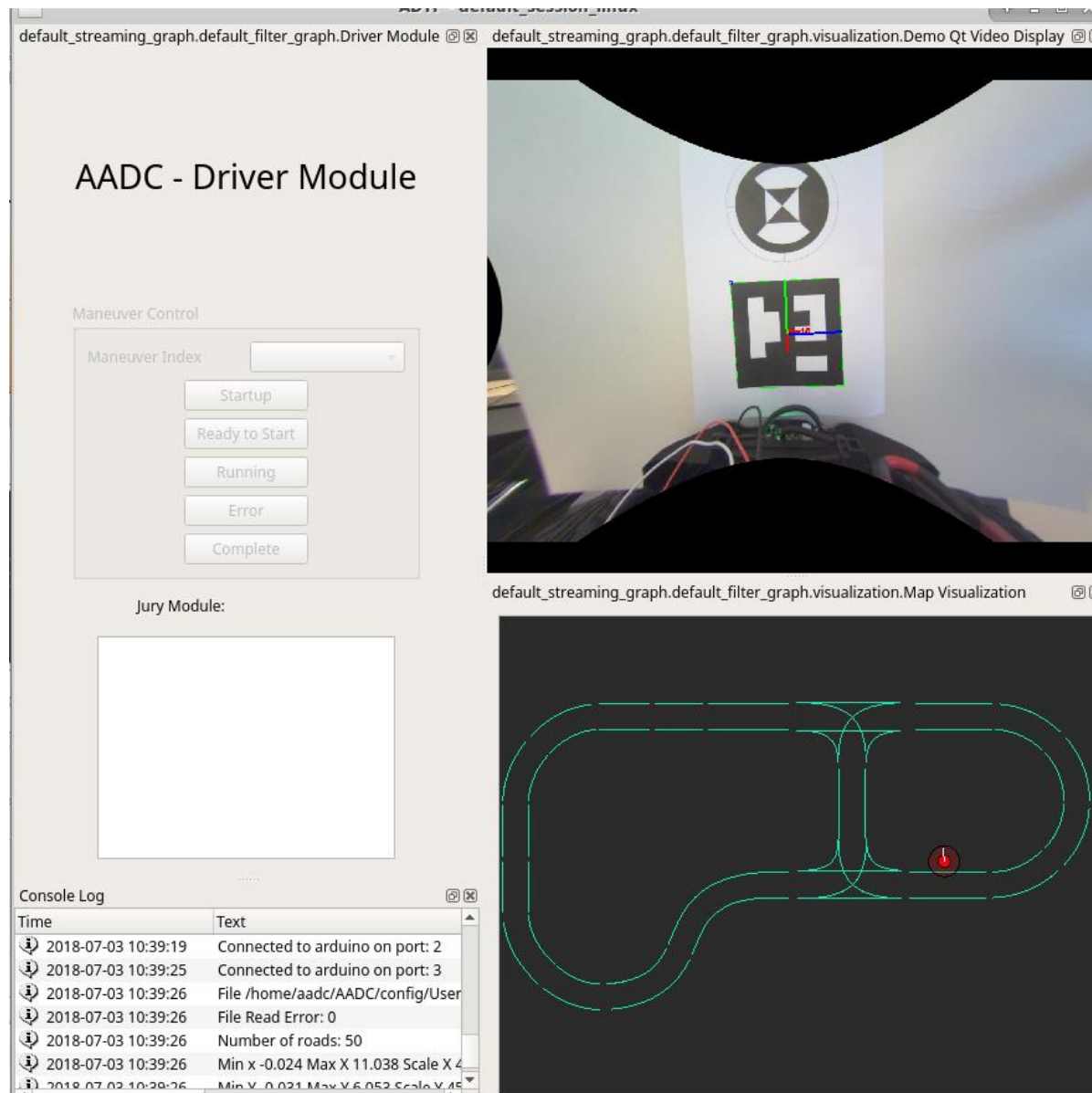
6.2.1.1 default_streaming_graph



6.2.1.2 default_filter_graph



When you start the session, the following user interface appears:



7 Jury Communication

During the test event and the competition itself the teams are not allowed to control the car by themselves. The AADC Jury will control the car with the following procedure to ensure that the car is really driving autonomously.

So it is strictly recommended for the Teams to work with the provided tools to ensure that the communication with the Jury is working properly during the competition.

7.1 General Procedure

1. Student Team starts their car. The used configuration should start automatically after booting and the configuration must include the Base filters and the Driver Module for the jury communication
2. Jury starts Jury Application
3. Jury connects to the student's car.
4. Jury loads the Maneuverlist in the Jury Module Module Application by pressing the "...” button in the GUI
5. Jury sends the Maneuverlist to the student's car.
6. Student's car should process the received maneuver list.
7. The car of the team should send a "isReady" state to the Jury
8. After the Jury Module has received the "isReady" state of the car it send the "Start" command to the car.
9. The car is now allowed to start going through the parcour.

7.1.1 Jury Application

This module will be used by the Jury to control the car in the test event and at the competition procedure. The binaries for Windows and Linux are located in the ADTF Source Package in folder

```
../_install/bin/application/juryApplication
```

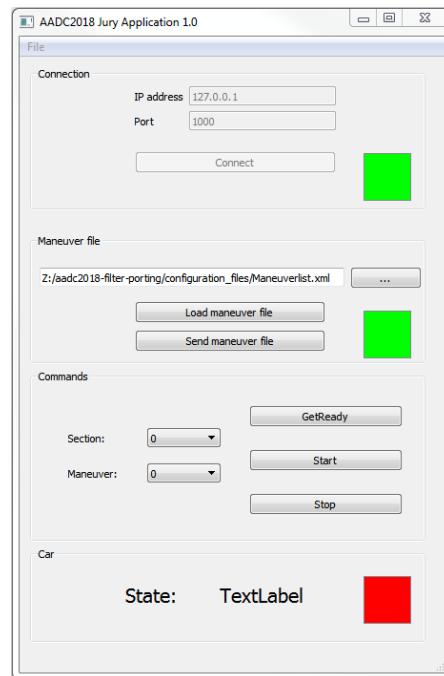


Figure 48 GUI Jury Application

7.1.2 Jury Actions

The jury sends its actions to the car by sending structs of type `tJuryStruct`.

```
typedef struct
{
    tInt16 i16ActionID;
    tInt16 i16ManeuverEntry;
} tJuryStruct;
```

The definition of the struct is contained in the file `/include/aadc_structs.h`

Valid actions are listed in the file `/include/aadc_jury.h` and are the following:

```
enum juryActions{
    action_stop=-1,
    action_getready=0,
    action_start=1
};
```

7.1.3 Car States

The car must send its state back to jury by using the following struct of type `tDriverStruct`:

```
typedef struct
{
    tInt16 i16StateID;
    tInt16 i16ManeuverEntry;
} tDriverStruct;
```

The definition of the struct is contained in the file `/include/aadc_structs.h`

Valid states are listed in the file /include/aadc_juryEnums.h and are the following:

```
enum stateCar{
    stateCar_error=-1,
    stateCar_ready=0,
    stateCar_running=1,
    stateCar_complete=2,
    stateCar_startup=-2
};
```


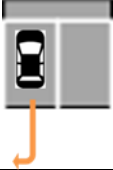
7.1.4 Sectors and Maneuvers

The whole course is divided into multiple sectors. Each sector contains one or more maneuvers. The sectors are used for rating and can be restarted during competition. The maneuvers are numbered globally over all sectors

7.1.5 Maneuverlist

The Maneuver List is a XML File concluding several sectors consisting of multiple maneuvers. This file defines what the vehicle has to do on streets, i.e. for instance to which direction it has to turn for the case of an intersection. There are no definitions of time. The file only defines the succession of actions.

The following maneuvers are defined for the competition:

Maneuver	Action
left	Turn left at the next crossing
Right	Turn right at the next crossing
straight	Keep straight at the next crossing
parallel_parking	Perform a parallel parking into the next parking lot (in driving direction right) (Remark: will not be requested in 2017 competition).
cross_parking <parking_id>	Perform a diagonal parking into the next parking lot (in driving direction right). <parking_id> is a unique parking slot id, as numbered in the provided map. It is passed with the “extra” field in the xml.
pull_out_left	Exit parking space to the left (from parking space, straight ahead looking in the direction of the street. In conjunction with cross parking only) 
pull_out_right	Exit parking space to the right (from parking space, straight ahead looking in the direction of the street) 
merge_left	Merge left into left lane

merge_right	Merge right into left lane (note: not used in 2018, placeholder)
-------------	--

A maneuver file can have the following content:

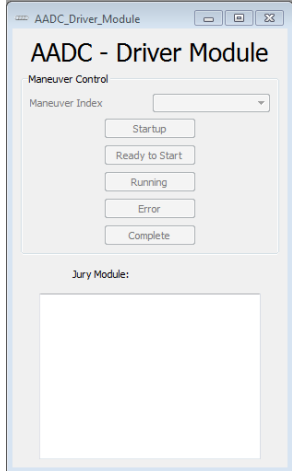
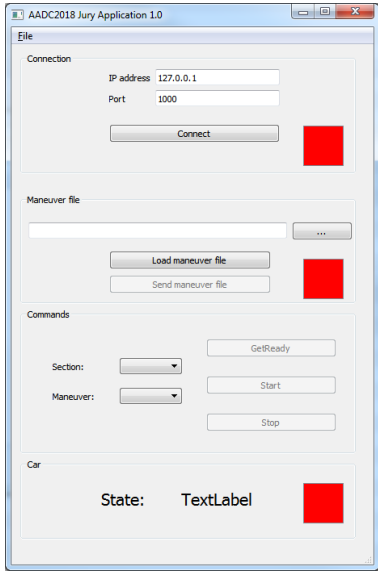
```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<AADC-Maneuver-List description="sample">
  <AADC-Sector id="0">
    <AADC-Maneuver id="0" action="left" />
    <AADC-Maneuver id="1" action="straight" />
    <AADC-Maneuver id="2" action="right" />
    <AADC-Maneuver id="3" action="straight" />
    <AADC-Maneuver id="4" action="straight" />
    <AADC-Maneuver id="5" action="left" />
  </AADC-Sector>
  <AADC-Sector id="1">
    <AADC-Maneuver id="6" action="straight" />
    <AADC-Maneuver id="7" action="straight" />
    <AADC-Maneuver id="8" action="straight" />
    <AADC-Maneuver id="9" action="straight" />
    <AADC-Maneuver id="10" action="straight" />
    <AADC-Maneuver id="11" action="cross_parking" extra="2"/>
  </AADC-Sector>
</AADC-Maneuver-List>
```

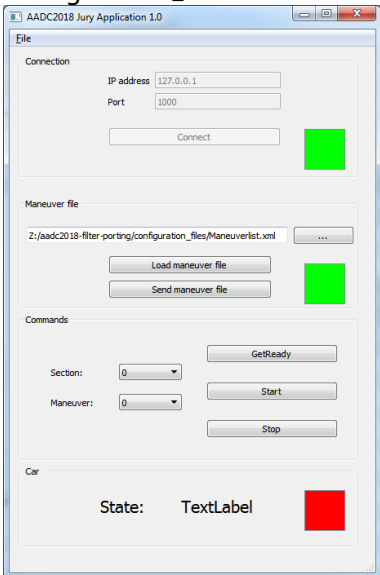
The maneuver list is loaded by the Jury with the Jury Module and not by the teams themselves. After loading the XML file in the Jury Module the maneuver list is transmitted via Ethernet to the student's car.

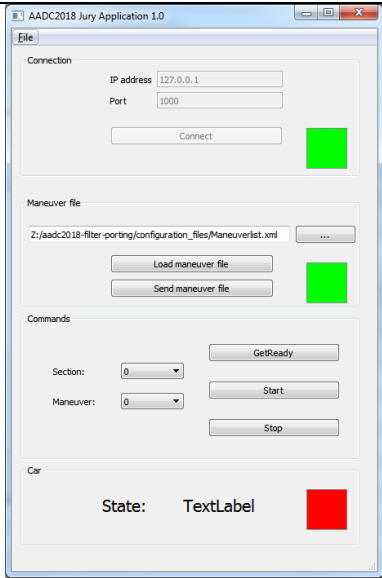
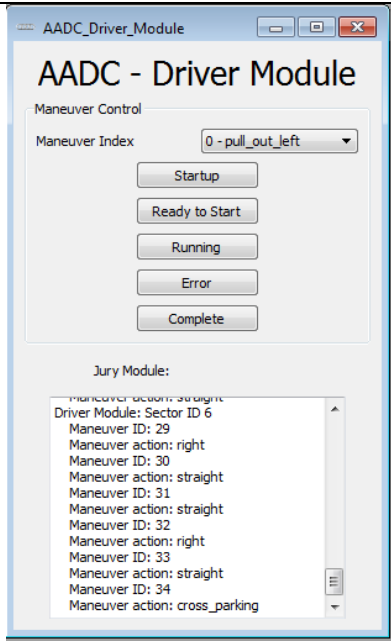
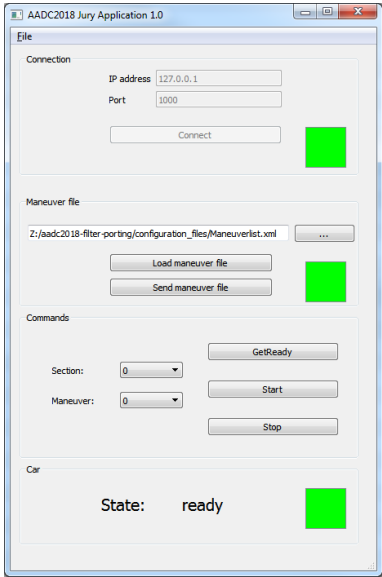
7.2 Example for Testing Jury Communication

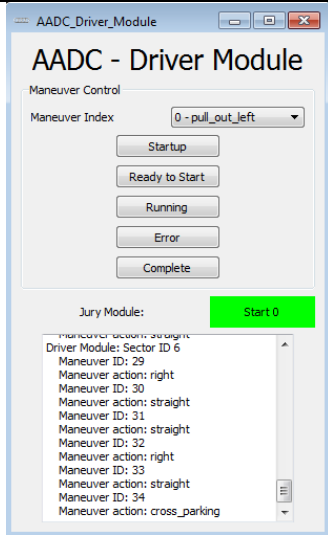
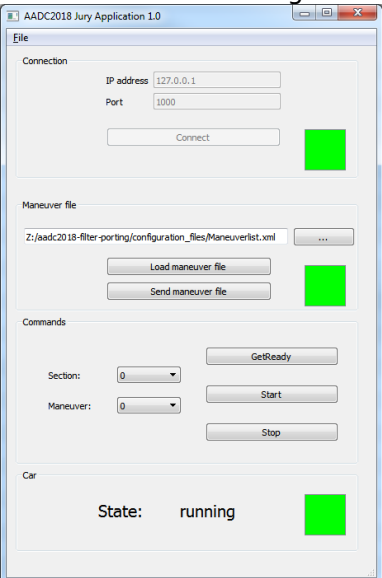
7.2.1 Starting the car

	Jury	Students car
1.	Setup a network with the car and another Jury PC. Make sure they can communicate with each other.	
2.		Open the configuration on the car

3.		<p>Run the Configuration and look for the window AADC_Driver_Module:</p>  <p>It should be empty at startup.</p>
4.	<p>Start the Jury Module application on the Jury PC</p> 	
5.	<p>Connect to the car by pressing the Button “connect”. The red state should change to green</p>	

		
6.	<p>Load maneuverfile (example is located in configuration_files\Maneuverlist.xml)</p> 	
7.	<p>Send the maneuverlist to the car by pressing the button “Maneuverlist”.</p>	<p>After receiving the maneuverlist from the jury the valid maneuvers are printed to the list in the window.</p>

		
8.		The car is now able to start the running of the course. It has to send the state "stateCar_READY". Within the example this can be done by pressing the button "Ready to Start".
9.	<p>The jury is now ready to start the student's car. The state is shown in the field "Driver State":</p> 	
10.	<p>The jury will now press the button "Start" with the maneuver index "0".</p>	The received jury action is shown in the window of the driver Module:

		 <p>The car should start now with the maneuver 0 (which is in sector 0)</p>
11.		<p>While running through the different maneuvers the car should send its state to the jury by always telling the current maneuver index. In the Driver Module this can be done by selecting the Maneuver Index from the dropdown and by pressing the button “Running”. The state “stateCar_COMPLETE” must be send at the end of the complete course, not at the end of sectors or maneuvers</p>
12.	<p>The jury is always able to see the state of the car and its incrementing maneuver indexes:</p>	

7.2.2 Stopping the car

If the car should be stopped while running the course to restart a sector the jury will send a stop command to the car. This can be done by pressing the button “Stop” in the Application. Then the team is allowed to place their car at the begin of the sector again. Afterwards the jury will do the same steps as shown in Starting the car 7.2.1 but with choosing the correct section before pressing the “Start” button.

7.2.3 Finishing the course

When the course is completely finished the car should send the state “stateCar_COMPLETE”.