Framework for Varied Sensor Perception in Virtual Prototypes

Stefan Mueller, Dennis Hospach, Joachim Gerlach, Oliver Bringmann, Wolfgang Rosenstiel
University of Tuebingen
Tuebingen
{stefan.mueller,dennis.hospach}@uni-tuebingen.de

Abstract

To achieve a high test coverage of Advanced Driver Assistance Systems, many different environmental conditions have to be tested. It is impossible to build test sets of all environmental combinations by recording real video data. Our approach eases the generation of test sets by using real on-road captures taken at normal conditions and applying computer-generated environmental variations to it. This paper presents an easily integrable framework that connects virtual prototypes with varying sensor perceptions. With this framework we propose a method to reduce the required amount of on-road captures used in the design and validation of vision-based Advanced Driver Assistance Systems and autonomous driving. This is done by modifying real video data through different filter chains. With this approach it is possible to simulate the behavior of the tested system under extreme conditions that rarely occur in reality. In this paper we present the current state of our virtual prototyping framework and the implemented plug-ins.

1. Introduction

In recent years advances in embedded systems and sensors technology have lead to a tight integration of the physical and digital world. Such systems, having connections to the physical world through sensors and actors and also having an embedded system for communication and processing, are often considered as Cyber-Physical Systems (CPS). These CPS are accompanied by new challenges in design and verification. Many examples for CPS can be found in a modern car, especially in the range of Advanced Driver Assistance Systems (ADAS), appearing more and more in the latest car generations. These ADAS heavily rely on sensor perception. The major problem is to guarantee functional safety requirements especially if ADAS are taking over more and more active control over the vehicle. These systems need to operate correctly in very different environmental conditions which are strongly influenced by the traffic situation, weather conditions, illumination, etc. This requires a high amount of on-road captures to test all combinations of environmental influences. Nevertheless, a total coverage is impossible. To address the issue of the amount of needed on-road captures, this paper presents an approach to reduce the number of captures by adding synthetic weather conditions to real captures. The presented framework allows to explore the virtual prototype, the Electrical/Electronic (E/E) architecture and the software running on it in the scope of many different use cases. In this manner it is possible to generate variations of environmental conditions of a traffic situation or add rarely occurring weather to existing on-road captures.

2. Related Work

The challenges in safety evaluation of automotive electronics using virtual prototypes are stated in [BBB+14]. With current validation methods, $10^7 - 10^8$ hours of on-road captures are needed to verify the functional safety of a highway pilot system in an ISO 26262 compliant way [Nor14]. Most vision-based ADAS techniques heavily rely on machine learning algorithms such as neural networks and support vector machines (SVM) as presented in [MBLAGJ+07, BZR+05] and/or Bayesian networks [BZR+05]. All these approaches have in common that they need to be trained with well selected training data [GLU12, SSSI11]. There are approaches to generate synthetic training data, where image degradation [ITI+07] or characteristics of the sensor and the optical system [HWLK07] are used to enlarge the training data. Most ADAS employ several sensors, networks and Electrical Control Units (ECU) to fulfill their work, which results in a complex scenario that can be considered as a cyber-physical system [Lee08]. A methodology to generate virtual prototypes from such large systems and keep a maintainable speed is shown in [MBED12]. It uses different abstraction levels to reach high performance of the controller and network models which are connected to a physical environment simulation. Another paper that covers virtual prototyping in the scope of ADAS comes from Reiter et al. [RPV+13]. They show how robustness and error tolerance of ADAS can be improved with error effect simulation. None of these works has presented a consistent way to connect the test and training methods to virtual prototypes.

3. Connecting Environment to Virtual Prototypes

To consider a virtual prototype under varying conditions, it is necessary to connect the virtual prototype to the environment. This connection is realized through a virtual sensor which models the properties of the corresponding real sensor. The advantage of using virtual sensors is that they can be applied in every design stage.
In the succeeding chapters we focus on vision-based virtual prototypes and refer to the following example applications:

- an in-house developed Traffic Sign Recognition (TSR) which is targetable to different hardware platforms. The Framework is used to train and verify the results of the integrated SVM.
- the Caltech Lane Detection Software (LD) which is presented in [Aly08]. We use this software as a second independent implementation of an ADAS algorithm to show the ease of integration of this framework into existing virtual prototypes.

As this work is focused on vision-based systems, the connection between virtual prototype and environment is done by a virtual camera. The input of the camera is an image of the environment. In our approach, this image is dynamically generated by modifying an existing video stream as shown in figure 1. These modifications may include different environmental impacts, which are sent to the virtual prototype and can on the one hand be used for detection of faults in hardware or software and on the other hand for fitness evaluation of the implemented algorithms.
The camera model and the dynamic generation of environment data is described in detail in the next chapter.

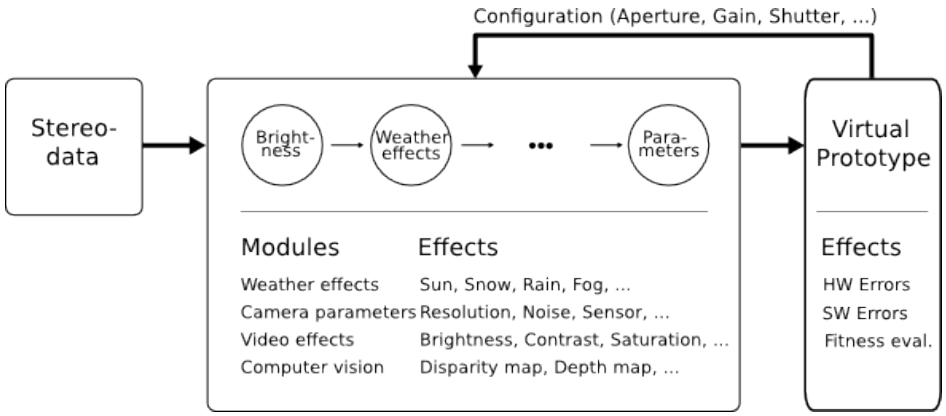

Figure 1: Parameters affecting the preparation chain

3.1. Data Acquisition with Virtual Sensors

The data acquisition with virtual sensors is different from the data acquisition of real sensors. The real environment provides all available information at a time, whereas virtual environment cannot deliver all this information simultaneously due to limited computing power. For example, a digital camera can vary in aperture, exposure time, gain, etc., which all influence the appearance of the captured image. To address this issue, the virtual sensor has to communicate with the virtual environment and request the desired values. The virtual environment then generates the desired result. The parameters which influence the requested value can be divided into two groups: sensor parameters and environment parameters. Examples for typical environment parameters are brightness, rain rate or the amount of fog. Examples for typical sensor parameters are given in figure 1. To ease the handling of the communication between sensor and environment, a generic, modular, plug-in based framework was created. It was designed under the following premise:

- generic data format and flexibility
- C/C++ interface because most simulation environments allow the call of C/C++ functions or can co-simulate it
- lightweight interface for data exchange
- easy to integrate in virtual prototypes

To achieve maximum flexibility, the framework uses three types of plug-ins for preparing the input data: source, filter and sink. These can be connected to preparation chains. Sources read a recorded scenario and pass it to the first filter or the sink. Filters modify the incoming data and pass it to the next stage, which can be a filter or a sink. There are two categories of sinks - online sinks and offline sinks. Both sink types provide the synchronization between one or more filter chains and the output destination. Online sinks are used for real-time capable processing chains and communicate directly with the prototype, where real-time is determined by the needs of the connected prototype. Let T_{cycle} be the time consumed by the prototype for processing one frame, N the number of filters in the chain, T_i the time that the i-th filter needs for processing and $T_{transport}$ the time consumed for the transport between the data generation and the processing unit of the prototype. The following

equation must hold if the system has to run in real-time:

$$T_{cycle} > \sum_{i=1}^{N} T_i + T_{transport}$$

Depending on the used abstraction level of the virtual prototype, the sink can be connected via different communication channels like a Transaction Level Modeling (TLM) interface or a network protocol. In contrast, offline sinks are used to prepare computationally intensive filtering jobs. Offline sinks store the received results for subsequent online runs. The preparation filter chain works with a pull mechanism, where the sink triggers the processing of each frame. This behavior is important for online sinks because it allows to work in sync with the simulation of the virtual prototype. Online sinks can also receive resulting data from the virtual prototype and can act as a source for an evaluation chain. This chain works with a push mechanism, which is important for systems with an asynchronous processing behavior, because it allows the evaluation to run on its own frequency.
The communication between these plug-ins is done by integration of the boost iostream library [ios]. By using the boost asio library [Koh] as a tunnel for the iostreams, it is possible to distribute the work over several computational resources. The generic data format is designed to abstract the data from all dependencies. This allows to build and run the framework on different platforms and the only library dependency that must be fulfilled by the simulation environment is the commonly used boost library. The communication between the different plug-ins is package-based. Each package has a type-id, which defines its content. This ensures the correctness of the chain. For example, a source that sends packages containing a rectified stereo image cannot be connected to a filter that expects packages containing a depth map.
To ease test creation and execution of such plug-in chains we created a Qt-based GUI which is designed to run independent of the server process. This allows to remote control the data preparation on computational resources. Prepared test cases can be stored as XML file and may be used to start the server process from the command line in batch processes.

3.2. Plug-in Chain for Environmental Variations

In the following we present the already implemented plug-ins which can be combined to chains to apply environmental variations on the input data. For illustration, figure 2 shows a full setup of a virtual prototype with preparation and evaluation chains in an online scenario.

3.2.1. Sources

By now we implemented image source plug-ins for image file sequences and video files. Both exist in a single and stereo image version. Each image source reads the specified data source using the ffmpeg library [ffm] and passes the data frame by frame to the succeeding plug-in. The only difference between single and stereo sources is that the stereo sources transfer two images per data package. Besides the normal image formats, there are sources for the images which are converted into scene radiance values after acquisition and for the output of the CarMaker software from IPG [Car]. The radiance value images are used by the plug-in which changes the sensor characteristics and the brightness. These are described later on.

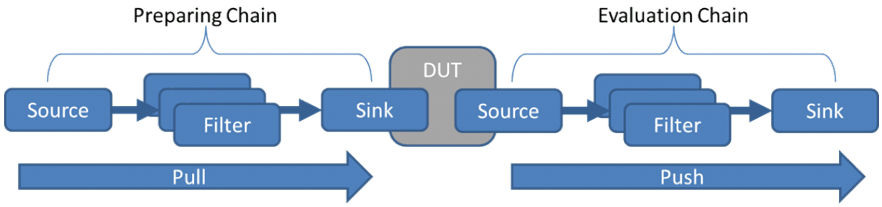

Figure 2: Work flow using preparation and evaluation chain in an online setup

3.2.2. Brightness

Probably the simplest variation to introduce to images is brightness. With real cameras, variations in brightness often occur due to the relatively slow adaption of the camera to changing incident illumination. An often-observed effect is over- or underexposure of the images. The reader may imagine driving into a tunnel or coming out of it. During the time the camera adapts the shutter and gain, all images will suffer from bad exposure.
To simulate this effect, we chose the following method: Saturating multiplication of the image in pixel space will lead to similar effects as described above. Let I denote a 2D image and $I(x, y)$ be the intensity value at position (x, y). Then this operation may be described as

$$I(x,y) = \begin{cases} a * I(x,y) & \text{if } a * I(x,y) < 255 \\ 255 & \text{else.} \end{cases}$$

where $a \in \mathbb{R}^+$. This leads to a spreading of the histogram with loss of information, where the pixel intensities run into saturation (see figure 3).
An alternative way of changing brightness is by doing it in the scene radiance space as presented in [HMB$^+$14]. Changing the incident scene radiance before the virtual prototype remaps it into pixel space is a more physically correct way of changing brightness and is the way to do it if one would like to simulate the virtual camera as an adapting system. The pixel values of the virtual camera will only run into saturation, if the parameters of the model are set accordingly.

3.2.3. Depth Information

For more complex variations like rain, fog or other effects that reduce the visibility, the image itself is not sufficient. Per pixel depth information is also necessary. This information can be sourced from stereo images. To convert the stereo images into a depth map, a chain of three plug-ins is used. The first filter generates a disparity map from the stereo images. This disparity map is generated by the Semi-Global Block Matching algorithm (StereoSGBM) of the OpenCV library [ope]. Afterwards, the following filter refines the disparity map by closing holes in the map. The holes in the disparity map are filled with the values of the neighbor pixels as discussed in [Sti13]. At last, the third filter calculates the depth information from the disparity map and supplies the left input image and the depth map to the succeeding filter.

3.2.4. Rain

The simulation of rain is a very complex task by itself and simulating every little aspect of it is quite impossible. The simulation of rain has mainly been addressed in the scope of computer vision,

aiming at generating visually convincing results. We have created a rain filter that is physically grounded, rendering rain streaks that follow the mathematical probability distributions of real rain. We are currently still evaluating the performance of the results with respect to the effects a sensor perceives.

3.2.5. Sensor characteristics

The virtual prototype acting as the connection of the physical world to the cyber-generated image heavily depends on the parameters of the simulation. The outcome can be significantly different when parameters of color channel sensitivity to illumination, color balance, noise characteristics at different gain levels or sensor size changes. To test the simulation with different optical parts of the virtual prototype, we developed a filter to map the image data from the camera system with which we recorded our sample data to another camera system. This virtual camera system can be based on a real camera, that has been calibrated against our recording system, or it could as well be a pure virtual system, exposing the possibility of simulating whichever parameter set is worthwhile testing. Color processing and color balance of a target system are also implemented. Further, we would like to be able to model the addition of sensor noise and possibly temperature drift of the sensor. Latter effects are under current development but need further evaluation.

3.2.6. Fog

For an initial rendering of particle effects like fog, there is a plug-in to pipe the data through the scene rendering tool Blender [ble]. This plug-in takes the scene image and transforms the image into a Blender scene according to the information from the depth map. Within this generated blender scene, various effects could be applied to the scene, rendered and returned to the next element in the chain. The current rendering of fog uses the standard Blender effect and is not evaluated in matters of its quality.

3.2.7. Sinks

As described earlier there are two kind of sinks. The offline sink just writes out the resulting images to a video stream, image sequence or to the display. The online sink transports the images to a co-simulated virtual prototype. This online sink establishes a bi-directional connection to the simulation and allows to transfer the images in sync with the simulation time. The online sink can also act as a source for an evaluation chain and return measures from the virtual prototype over the bi-directional connection. These received measures can be evaluated in several ways and lead to a new parameterization of the preparation chain for a new test run and is intended to build up a parameter space exploration.

3.3. Integration

In order to include all the aspects of our TSR, we use SystemC to model the microcontroller, the memory hierarchy and the on-chip busses as well as the automotive-networks like MOST, FlexRay and CAN. This is used to evaluate robustness of the embedded software with respect to the entire chain of effects from the sensor via the E/E architecture to the ECU architecture under varying environment conditions. As proof of the ease of integration into third-party prototypes we

Figure 3: Brightness variations

Figure 4: Scene at normal environmental conditions

connected our system to the Caltech Lane Detection Software [Aly08]. This software is written in C++ and uses the OpenCV library. The basic integration of this software to our framework took about 1 hour and needed 25 lines of code for the communication. The conversion of the images delivered by the framework into the OpenCV format required 18 lines of code.

4. Results

In this chapter we present first promising pictures generated by this framework. In figure 3 several brightness variations are shown. Figure 4 shows the original scene whereas figure 5 shows the same scene with the modification of rain at a rate of $30.0\ mm/hr$. Brightness was left unchanged. The stereo images used for the results are captured with a camera, which consists of three image sensors with a maximum resolution of 1280x960 pixels. In total, over 330 km of city and interurban on-road captures have been acquired.
The used SVM-based TSR system can discriminate between 12 classes of speed signs. For the evaluation three test sets are used. A route at bright weather (track 1), the same route at rainy weather (track 2), and a different route with many road signs at a diffuse light (track 3) for training purposes. The initial training of the TSR was done with track 3 and used to evaluate track 1

Figure 5: Image with artificially added rainfall (rain rate: $30.0\ mm/hr$) and unchanged brightness

to show that the training works for the chosen route. Then track 2 is evaluated to measure its performance on rainy conditions. After that we modified track 3 by applying different rain rates like in figure 5 and used it to enhance the training of the TSR. The newly trained TSR is used to evaluate track 2 again. The recognition results are shown in table 1. The difference in the number of total recognitions between track 1 and track 2 is caused by two factors: the driven velocities and the acutance. The test routes were driven at different velocities and therefore a traffic sign may be visible in more or less frames. The acutance is important for the circle detection. Rain adds some more or less heavy blur to the images, so that more parts of the image are detected as circles. The currently used TSR application does not do any kind of circle aggregation previous to the classification step.
Comparing the performance of both trained TSRs on the rainy day test set (track 2) shows that the number of correct recognitions rises from 44.8 % to 64.2%. Even though the result is not as high as for the bright day (track 1), it has increased significantly.

Test set	Training set	True	False	Sum	Percent
Bright day(track 1)	track 3	52	11	63	82.5%
Rainy day (track 2)	track 3	43	53	96	44.8%
Rainy day (track 2)	track 3 with add. rain	61	34	95	64.2%

Table 1: Evaluation results

5. Conclusion

In this paper, we introduced a platform independent framework for simulation of environmental conditions and the use of virtual prototypes to evaluate their effect on the embedded software and the underlying E/E architecture. The platform is based only on free and open C/C++ libraries: boost::iostreams, boost::asio, OpenCV and ffmpeg. Due to the flexible streaming concept, we are able to add various effects to video material and thus simulate many different combinations of environmental effects and sensor characteristics of the targeted optical system. This allows us to generate many different training sets from the same on-road captures. Furthermore, our system supports the requirement to evaluate different environmental conditions on a given trajectory to compare efficiency and effectivity of different algorithms for specific ADAS problems. An evaluation step may then rate the overall performance of the virtual prototype and give feedback to conduct automatic parameter space explorations on a higher level of accuracy.

6. Future work

The next steps will be to validate the quality of each synthetic environmental effect related to the sensor perception and the requirements of the different ADAS applications in more detail. For an automatic parameter space exploration different search algorithms will be implemented to speed up the search of specific functional limits of the system.

Acknowledgment

This work was partially funded by the State of Baden Wuerttemberg, Germany, Ministry of Science, Research and Arts within the scope of Cooperative Research Training Group and has been partially supported by the German Ministry of Science and Education (BMBF) in the project EffektiV under grant 01IS13022.

References

[Aly08] Aly, M.: *Real time detection of lane markers in urban streets*. 2008 IEEE Intelligent Vehicles Symposium, June 2008.

[BBB+14] Bannow, N., M. Becker, O. Bringmann, A. Burger, M. Chaari, S. Chakraborty, *et al.*: *Safety Evaluation of Automotive Electronics Using Virtual Prototypes: State of the Art and Research Challenges*. In *Design Automation Conference*, 2014.

[ble] *blender.org - Home of the Blender project - Free and Open 3D Creation Software*. `http://www.blender.org/`, visited on 23/05/14.

[BZR+05] Bahlmann, Claus, Ying Zhu, Visvanathan Ramesh, Martin Pellkofer, and Thorsten Koehler: *A system for traffic sign detection, tracking, and recognition using color, shape, and motion information*. In *IEEE Intelligent Vehicles Symposium (IV 2005)*, 2005.

[Car] *IPG: CarMaker*. http://ipg.de/simulationsolutions/carmaker/, visited on 23/05/14.

[ffm] *FFmpeg*. http://www.ffmpeg.org/, visited on 21/05/14.

[GLU12] Geiger, Andreas, Philip Lenz, and Raquel Urtasun: *Are we ready for autonomous driving? the kitti vision benchmark suite*. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[HMB+14] Hospach, D., S. Mueller, O. Bringmann, J. Gerlach, and W. Rosenstiel: *Simulation and evaluation of sensor characteristics in vision based advanced driver assistance systems*. In *Intelligent Transportation Systems, 2014 IEEE 17th International Conference on*, pages 2610–2615, Oct 2014.

[HWLK07] Hoessler, Hélène, Christian Wöhler, Frank Lindner, and Ulrich Kreßel: *Classifier training based on synthetically generated samples*. In *The 5th International Conference on Computer Vision Systems*, 2007, ISBN 9783000209338.

[ios] *The Boost Iostreams Library*. http://www.boost.org/doc/libs/1_54_0/libs/iostreams/doc/, visited on 22.08.2013.

[ITI+07] Ishida, H., T. Takahashi, I. Ide, Y. Mekada, and H. Murase: *Generation of Training Data by Degradation Models for Traffic Sign Symbol Recognition*. IEICE Transactions on Information and Systems, pages 1134–1141, 2007.

[Koh] Kohlhoff, C.: *Boost.Asio*. http://www.boost.org/doc/libs/1_54_0/doc/html/boost_asio.html, visited on 22.08.2013.

[Lee08] Lee, Edward A.: *Cyber physical systems: Design challenges*. Technical report, EECS Department, University of California, Berkeley, Jan 2008.

[MBED12] Mueller, W., M. Becker, A. Elfeky, and A. DiPasquale: *Virtual prototyping of cyber-physical systems*. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 219–226, Jan 2012.

[MBLAGJ+07] Maldonado-Bascon, S., S. Lafuente-Arroyo, P. Gil-Jimenez, H. Gomez-Moreno, and F. Lopez-Ferreras: *Road-Sign Detection and Recognition Based on Support Vector Machines*. IEEE Transactions on Intelligent Transportation Systems, 8(2):264–278, June 2007, ISSN 1524-9050.

[Nor14] Nordbusch, Stefan, Robert Bosch GmbH: *Vision or Reality – The Way to Fully Automated Driving*, 2014. https://www.edacentrum.de/veranstaltungen/edaforum/2014/programm.

[ope] *OpenCV*. http://opencv.org/, visited on 06/09/13.

[RPV+13] Reiter, S., M. Pressler, A. Viehl, O. Bringmann, and W. Rosenstiel: *Reliability assessment of safety-relevant automotive systems in a model-based design flow*. In *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*, pages 417–422, Jan 2013.

[SSSI11] Stallkamp, J., M. Schlipsing, J. Salmen, and C. Igel: *The german traffic sign recognition benchmark: A multi-class classification competition*. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1453–1460, July 2011.

[Sti13] Stickel, Christoph: *Simulation und Modellierung von Witterungsbedingungen in der Auswertung videobasierter Umfeldsensorik*. 2013.