Developing autonomous vehicle research platform – a case study

K Belcarz  T Białek  M Komorkiewicz and P Żołnierczyk
FEV Polska sp. z o. o., Cholerzyn 467, 32-060 Liszki
E-mail: komorkiewicz@fev.com

Abstract. The article presents the case study of autonomous vehicle research platform development. All the important aspects of car conversion are presented. Firstly the actuation from manual to automatic acceleration, braking and steering is discussed. Secondly the Nvidia Drive PX 2 GPU based embedded platform chosen as the control unit as well as the ROS software are also described. Thirdly all the sensors that are needed to allow the car to sense the environment around (cameras, radar, LIDAR, precise GPS) and react to any dangerous situation are presented. Finally the vision algorithms that were developed based on the designed autonomous vehicle research platform are described.

1. Introduction

Nowadays, thanks to companies like Tesla or Uber, autonomous vehicles have become almost a reality. On the other hand the question about the level of this autonomy needs to be asked. According to the well-known SAE (Society of Automotive Engineers) vehicle levels of autonomy [1] the cars that are on the market can be graded as level 1-3, which means that even the most sophisticated cars are able to operate autonomously only in limited number of situations (e.g. highways, traffic jams etc.) and require the driver to be ready to take over in every moment (driver is still 100% responsible).

The problem with this approach is that people often misuse the provided technology. It is partially caused by the fact that, after several successful journeys, the driver loses interest in paying attention to the task of monitoring the autonomous driving mode of the vehicle and starts doing something else. This might result in a distraction which may cause the driver to miss the 1% of critical situations. The solution to this problem is reaching the next levels of autonomy (4-5) where driver interaction will not be required.

It is obvious that removing the driver completely from the control path is not an easy task and requires an enormous amount of work and completely new algorithms and sensors. It can be observed that research and development activities conducted by the biggest players in the automotive industry are mainly performed in two ways: by using advanced computer based simulators (e.g. Carla [2]) and by extensive use of test vehicles research fleet. As not everything can be tested in a simulated environment, smaller companies and universities also face a need to have the autonomous vehicle research platform to be able to develop the new technology.

The first idea would be to buy an existing level 2-3 car and turn it into a research platform. Such approach is not the best way to go due to high costs of available cars as well as the problem with taking control over the vehicle actuators. Manufacturers are not willing to share the know-how about sensors and controls mainly due to the cybersecurity issues (the less people know, the harder it is to hack the car) and also to block the competition from easy access to its controls strategies.
The second approach is to buy an ordinary car without autonomous features and add the sensors and drive-by-wire controller on one’s own. In this article a case study of such a conversion (Figure 1) is described as well as the vision algorithms developed based on this platform are presented.

![Figure 1. The fleet of FEV Smart Vehicle Development demonstrator cars](image)

The described work was part of larger Smart Vehicle Development program conducted internally by the FEV group around the world (FEV GmbH, FEV NA, FEV Turkey, FEV Poland) [3].

2. Car choice and modification

Adding a drive by wire system to a pure gasoline/diesel car is not an easy task. On the one hand it is because the mentioned problems with getting information about CAN bus matrix, on the other it is because most of the cars still have the mechanical gear/pedals. Taking control over steering is relatively easy due to the fact that almost all new cars have power assisted steering. There is a group of vehicles that is electrified (hybrids or pure electric) that are much better suited for being used as autonomous vehicle research platform. This is because in most of such cars pedals and gear changing mechanisms are usually implemented as a drive-by-wire system. Lack of mechanical actuators simplifies a lot the process of taking control over the vehicle steering, breaking and acceleration with limited number of additional electronic control units required.

Unfortunately it is not as simple as it may seem. It is important to remember that introducing changes into the most critical subsystems in the car may be very dangerous and such a vehicle requires additional certification after modifications are introduced.

For our program the Ford platform (Ford Mondeo/Fusion) was chosen which is widely used by a lot of companies (e.g. Uber, Nvidia etc.). It is so popular due to a well performing hybrid powertrain as well as the ability to control the automatic transmission gears (D, L, N, R, P) as it has knob rather than a lever. The actuators control solution was identified and implemented by FEV NA.

3. Central control unit and ROS

Choosing the right SW/HW architecture for the autonomous vehicle research platform is not an easy task. There are multiple possibilities which on one side affect the ease of algorithm development on the other impact the final quality of the designed system (safety, automotive grade solution, etc.). Another important factor that must also be taken into consideration is the power needed to run the system and the heat that needs to be dissipated in the trunk. For most passenger vehicles the de facto standard is a 2kW power supply.

A lot of the solutions (e.g. IDiada, AiMotive) are based on the standard or modified PC. This is a good and relatively low cost approach which allows fast development of the algorithms and ensures scalability of computing performance as well as storage space. The drawback is problematic sensor and vehicle integration (e.g. video I/O, CAN busses, etc.) as well as meeting real-time requirements (with a standard OS). The other solution is a dedicated embedded controller (e.g. DSpace MicroAutoBox, Nvidia DrivePx 2 [4,5]) with additional HW accelerators. The cost of such systems
and learning curve is much higher but the solution is more like automotive grade and allows good integration with all the sensors in the car.

It was decided by FEV to use a dedicated research platform based on ARM architecture and GPU accelerators for deep learning algorithms execution. The research indicated that the optimal solution is the Nvidia Drive PX 2 platform (Figure 2). It is based on automotive grade Aurix processor and two very advanced Parker SoCs (system on chip) interconnected with 10Gbps network switch as presented in Figure 3.

Figure 2. The Drive PX 2 hardware (NVidia materials)

Figure 3. The Drive PX2 architecture (NVidia materials)
Each SoC device encapsulates 4x ARM Cortex A57, 2x Nvidia Denver2 CPU one Pascal GPU with the unified memory addressing and one additional discrete GPU (separate memory). In total there are 16 separate computing devices on the Drive PX2 platform enabling very efficient distribution of computing tasks among them. The platform has CAN, Broad Reach, Ethernet and GMSL (Gigabit Multimedia Serial Link) for video I/O.

After choosing the HW platform the next important decision was the OS (operating system) and algorithm design framework. There are again multiple options of OS such as Windows ARM, Linux or QNX. Since the first one is not supported on the Drive PX 2 platform, the only remaining options were Linux or QNX. Linux is open source and free but not real-time; the QNX system is the opposite – non-free but real-time. It was decided to use the Linux option as the embedded developers are quite well familiar with this OS.

As Linux is only an operating system, the Robot Operating System [6] (designed by Willow Garage and Stanford Artificial Intelligence Laboratory) was chosen as the next layer of architectural abstraction. It allows to create nodes which can communicate with each other by exchanging messages using the Ethernet network. The ROS is a centralized concept with one master and nodes which can connect and subscribe/publish messages. It is a widely used framework which allows to nicely setup the final system architecture out of a library of available packages. The concept was to design separate nodes for each sensor as well as for control algorithms and actuators. Using ROS allowed every global FEV site engaged in the project to develop a particular node and test it separately. It also enabled everyone involved to have an intuitive integration into the final solution. The overall architecture of the system with the ROS nodes is presented in Figure 4.

**Figure 4.** ROS node based system architecture

It is also worth to mention that the ROS has a large community of users and a lot of freely available packages such as 3D scene visualization software or LIDAR point cloud libraries.

4. Sensors
There are multiple sensors that can be used in autonomous vehicles to sense the surrounding and position the car in the environment, among them, the most commonly used are:

- Cameras
- Radars
- Ultrasonic
Some of the researchers argue that since a human is able to drive the car only with its eyes, then the cameras should be enough for the AI system to drive. Unfortunately this assumption was negatively verified by some dangerous accidents. This is because the camera based vision system is affected by the same impediments that limits the human eye, the most notable ones are for example darkness, direct sunlight, or fog.

It is a general consensus that the more versatile sensors are used the more robust the car is to changing environmental conditions. The next problem, which is not as trivial as it seems to be, is where to place the sensors around the car to ensure the best detection quality and smallest blind spots. The most important factors that must be taken into consideration are:

- Integration with existing elements of the car (bumpers, mirrors etc.)
- Sensors placement should not interfere with each other (e.g. LIDAR with GPS antennas, etc.)
- The sensors must be protected from environmental conditions as much as possible (rain, dust, dirt, snow, etc.)

Especially the last bullet point is worth to be mentioned. It is not a coincidence that the majority of autonomous cars are tested in California. It is due to long periods of very good weather condition. In Central Europe the weather is not so good which does not allow to mount the cameras outside the car without special cleaning equipment or considerations of how to eliminate the negative impacts that snow or dirt has on some sensors.

The following sensors were chosen for the FEV Smart Vehicle demonstrator:

- 360 degree LIDAR (Velodyne VLP16),
- 6 x wide angle 2 MP cameras (120 degrees),
- 2 x 2 MP front looking camera (60 degrees) for stereovision,
- 1 x narrow angle 2MP front looking camera (43 degrees),
- Industrial grade front facing Radar (> 70 GHz wave length),
- Precise two antennas dGPS with RTK and IMU correction (accuracy up to 1.5 cm)

The car setup and sensor mounting positions are presented in Figure 5. 3D printing technique was extensively used for creating the sensor cases/mounting stands.
5. Vision algorithms

5.1. Object detection

The object detection and classification designed by FEV is based on the deep learning algorithm utilizing convolutional neural networks (CNN) [7]. It is currently the most widely used machine learning technique that is able to deliver high quality results. The networks are very big (often more than >20M parameters) and require vast amount of computing performance. The CNN are very well suited to be efficiently trained and run on modern high-end GPUs.

One of the most important steps in machine learning is data collection and annotation. Unfortunately the authors were not able to find the annotated database of Polish roads so almost 2 TB of movies were collected using the car sensors. For traffic sign detection the 15,000 image frames were manually annotated and pedestrian detection required almost 20,000 manually annotated image frames.

There is no vision based algorithm that is able to correctly detect objects an all frames. That is why the object tracker algorithm was added, which is able to track lost objects on a few consecutive frames even if the detector was not able to work correctly. The bounding box on the image is reporting only pixel coordinates of the object, in order to use this information in autonomous controller, the object position must be reported in real world coordinates. Additional algorithms were created to calibrate the camera and measure distance to the objects.

The working system is presented in Figure 6. The FEV TSR (Traffic Sign Recognition) classifier is presented in Figure 7.

![Figure 6. The example result of the object detection algorithm](image-url)
5.2. Surround vision

Nowadays the bird eye view from surround cameras is a function present in most of the premium cars. It is a very good feature for supporting maneuvers like parking, but the view horizon is very short and ends usually about 1~2 m around the car. There is another option which is a spherical image projection [8]. It is a transform that allows to nicely visualize both the near and far objects around the car. In our program we have decided to implement this kind of surround visualization.

In geometry this technique is called stereographic projection. Concisely speaking it allows to map a sphere onto a plane. The projection is defined on the entire sphere, except for the projection point. It’s not possible to map a sphere onto a plane without some sort of distortion. The stereographic projection preserves angles and distorts areas. Navigation is mostly performed with angles which is the reason it is an acceptable trade-off.

The result is obtained by projecting points Q on the surface of the sphere from the sphere’s north pole N to point P in a plane tangent to the south pole S. Referenced points and axes orientation are presented in Figure 8. In such a projection great circles are mapped into circles, while loxodromes are mapped into logarithmic spirals. Depending on relative position of the projection plane and the sphere, z-axis in particular, a variety of equivalent transformation formulas can be formulated.

![Figure 8. Stereographic projection](image1.png)

![Figure 9. ZX plane projection](image2.png)

The point S is taken to have latitude and longitude \((\lambda, \phi)\)=\((-90,0)\), if \(\lambda_0\) as the central longitude and \(\phi_1\) as the central latitude, the transformation equations for a sphere of radius \(R\) are given by:
\[ u = k \cos(\phi) \sin(\lambda - \lambda_0) \]  
\[ v = k [\cos(\phi_1) \sin(\phi) - \sin(\phi_1) \cos(\phi) \cos(\lambda - \lambda_0)] \]

Where:
\[ k = \frac{2R}{1 + \sin(\phi_1) \sin(\phi) + \cos(\phi_1) \cos(\phi) \cos(\lambda - \lambda_0)} \]

\( k \) determines where the projected plane is located.

In order to depict how specific points are mapped, a ZX plane projection is presented in Figure 9. The distortion of the image heavily depends on projection plane placement. In this case upper hemisphere exhibits the most significant distortion. The distortion gets more extreme the closer the point on the sphere is to the origin of projection (north pole). This view also shows that stereographic projection is a bijective operation.

The first step of the FEV algorithm is to capture the image from all six (6) wide angle cameras and merge it into one big panoramic image [9]. The standard approach requires a manual or automatic calibration based on corresponding points and computing the image projection matrices. The second step is to compute the mapping from the 2D panoramic image into the surround view in polar coordinates.

Processing all six (6) images requires the computing performance that even on the Intel Core i7 3.6 GHz CPU takes more than 500 ms. To speed up the computation, the transformations done in the image processing path were merged together. Thanks to this approach, not all pixels must be
processed during panorama computation which saves a lot of operations. In order to implement this algorithm, the mappings were precomputed and stored in a LUT (look up table). The final solution was ported from CPU to GPU where the code was successfully parallelized and resulted in a 17 ms computing time. The speedup was about 29 times. The example panoramic surround view is presented in Figure 10.

6. Conclusion
Using the methodology described in this article, the FEV converted three Ford Mondeos into research platforms for its Smart Vehicle Development program. The fleet was used as the on-wheel laboratories for development of various controls (path following, adaptive cruise control, line keep assist, etc.) as well as for sensor data processing (Camera, Radar, LIDAR) algorithms. The embedded computing platform allowed to execute several neural networks as well as image processing algorithms in real time (>20 fps) with up to 8 cameras connected in the same time. The overall power consumption of the controller was below 300W. The implemented 360 degree surround vision algorithm improved driver situational awareness compared to commonly used bird eye view algorithm. FEV is currently working on the next generation autonomous vehicle research platform where the current vehicles are enhanced with additional capabilities and features.

References
[1] Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles, Surface Vehicle Recommended practice J3016 September 2016
[2] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, Vladlen Koltun CARLA: An Open Urban Driving Simulator, 1st Conference on Robot Learning 2017
[3] Alza‘bi, Hamzeh & Dwyer, Brian & Nagaraj, Sarika & Pischinger, Martin & Quail, Alanna. (2018). Cost Effective Automotive Platform for ADAS and Autonomous Development. Strite S and Morkoc H 1992 J. Vac. Sci. Technol. B 10 1237
[4] dSPACE MicroAutoBox2 Hardware accessed on 26.04.2018 from https://www.dspace.com/en/pub/home/products/hw/micautob/microautobox2.cfm
[5] NVIDIA DRIVE 5.0 Linux SDK, Development Guide, 5.0.5.0 Release, acces on 26.04.2018 from https://docs.nvidia.com/drive
[6] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y. Ng ROS: an open-source Robot Operating System, ICRA Workshop on Open Source Software 2009
[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1
[8] Wang, Benzhang & Tang, Fan & Liu, Hengzhu. (2018). Spherical Panorama Stitching Based on Feature Matching and Optical Flow. 83-88. Lecture notes in Engineering, Chinese Intelligent Systems Conference 2018
[9] Adel, Ebtsam & Elmogy, Mohammed & M. El-Bakry, Hazem. (2014). Image Stitching based on Feature Extraction Techniques: A Survey. International Journal of Computer Applications. 99. 1-8.