EasyChair Preprint

№ 1680

An End-to-End solution to Autonomous Driving based on Xilinx FPGA

Tianze Wu, Weiyi Liu and Yongwei Jin

October 16, 2019

An End-to-End solution to Autonomous Driving based on Xilinx FPGA

1st Tianze Wu
Insititute Of Computing Technology Chinese Academy Of Sciences
University of Chinese Academy of Sciences
Beijing, China
wutianze@ict.ac.cn

2nd Weiyi Liu
Shanghai JiaoTong University
Shanghai, China
liuweiyi@sjtu.edu.cn

3rd Yongwei Jin
Xidian University
Xian, China
yongweijin@outlook.com

***Abstract*—Nowadays, the autonomous driving topic is very hot, many people are trying to provide a solution to this problem. This time we build our own auto-driving car based on Xilinx Pynq-Z2, it provides an end-to-end solution which inputs images from camera and outputs control instructions directly. The platform also uses the power of Deep learning Processing Unit(DPU) to accelerate the inference process and provides a simulator for training and testing in virtual environment. If the car meets some situations which cannot be handled by AI model, it's easy to add extra traditional computer vision functions to our control system. So our platform can help people who want to try autonomous driving build their own model and test it efficiently. We hope that our platform can be easy to use and extend.**

***Index Terms*—Autonomous Driving, Machine Learning, Pynq-Z2, Field programmable gate arrays, Deep Learning Processing Unit**

I. Introduction

Autonomous Driving is getting more and more attention recently. The main problem to be solved in this area is how to meet the need of real-time computing. Though we now have many customed chips like GPU [1] and TPU [2] for machine learning acceleration, they are not able to provide both high performance and low power consumption. Also they are not flexible enough to adapt to different scenes.

To meet these requests, FPGA [3] is a good solution since it can provide high performance and low power consumption, most important, people can build accelerators for specified algorithms easily.

This time we build our own auto-driving car based on Xilinx Pynq-Z2 [4], it provides an end-to-end solution to autonomous driving and it uses the power of DPU [5] to accelerate the computing. Our goal is to build a general and easy-to-use platform for those who want to try their own ideas about auto-driving.

We find that in previous FPT competitions [6], many competitors use traditional computer vision methods or machine learning methods to detect objects and then make decisions based on the results [7] [8]. Our solution is quite different from theirs since we use a simple CNN model as our AI network and feed the model with pictures taken by the camera in the car and the model returns the control orders. The platform we use is Xilinx's Pynq-Z2 board, there is a DPU IP in its FPGA, with DPU's help, we can accelerate the inference process of the model and make it possible to run AI inference task in limited resource platform like Pynq-Z2.

We also build a simulator based on sdsandbox [9] using Unity3d [10], we can collect training data, test the model's performance and pre-train the model in the simulator. It highly increases the efficiency and accelerates the project development.

The rest of this paper is organized as follows. Section II describes the overview of our hardware structure and development environment. Section III presents the software architecture and details of methods we are going to use for the competition. Finally, the paper mentions the current progress and makes an conclusion.

II. Overview of our car

A. Hardware structure

The Fig. 1 is our robot car's photograph. Its body is made of acrylic board. There is one Pynq-Z2 board as the controller, it has a DPU IP for acceleration, the sensors we use now contains a camera only. The car has one capstan motor and one electric steering engine. Two rear wheels provide power while two front wheels provide steering capacity. We use two batteries to provide power, one for the Pynq board and one for the motors.

B. Development environment

The operating system we use in the car is created by PetaLinux [11], it has opencv and dpu support. We control the motors by directly writing values to physical memory address of FPGA, the drivers are implemented in FPGA. Most of our AI inference computing tasks are run by DPU, the tasks which DPU doesn't support are run by arm core embedded in Pynq-Z2. The driving decisions are made by AI model while the central controller is run in arm core.

We provide two language implementations for collect-data module, you can collect data for training in c++ or python environment. The c++ version is more commonly used while the python version has a good support in Pynq-Z2. The auto-run module is implemented in c++ since the DNNDK-v3.0 [12] version has only APIs for c++ language.

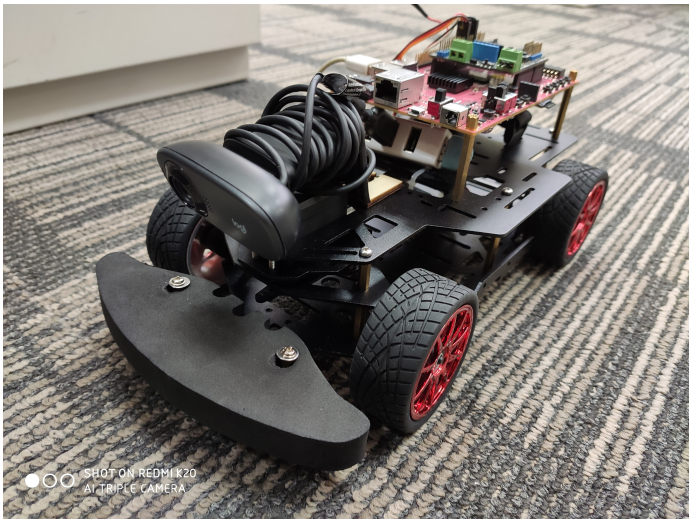

Fig. 1. Car overview.

III. Software Implementation and Algorithms

In most tasks, our solution can completely rely on machine learning methods, such as left-side driving, obstacle avoidance and crossroad driving. When it comes to other scenes, we use traditional computer vision algorithms and state machine to help us finish the task. The car only uses a camera as the sensor now, we may add some more sensors if a camera is not enough.

A. Software architecture

In our design, we aim to make the platform be easy to use and expand. There are three main modules in our system.

- The first part is Data Collect module. It needs human to control the car to finish some driving tasks. The program will record the images taken and real-time commands during the process. These data will be used as training data.
- The second part is PC Host module. Since our edge devices cannot provide huge compution power, we will use more powerful devices like normal computers to do the machine learning job. You can use any hardware you like to do the training and then use DNNDK provided by Xilinx to do some optimizations. DNNDK kit can do pruning, quantization, compilation and optimization to the trained model. After this step, we will have DPU kernels that can help us accelerate inference process in the car.
- The third part is Autonomous Driving module, it acts like the first module but the human controller of the car is replaced by machine learning network.

Since these three parts are separated, so it's easy to modify one module without affecting others. It will be easy to build a different neural network or use different hardware devices. We are also working on an virtual simulator for our platform, once it is finished you can train or evaluate the model based on virtual environment. This can help people build or test their model structure efficiently.

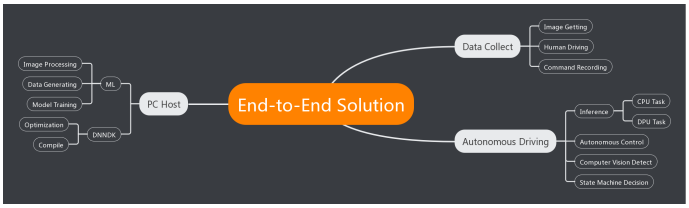

Fig. 2. Software Architecture

B. Work flow

Fig. 3 shows the control flow of our autonomous driving. The car can run in two mode, one is that the car will be in total control of machine learning methods, the other means that the traditional computer vision methods will be used when the car runs into the situation in which neural network is not able to handle.

The whole system will have one worker for running computer vision methods like objects recognition, one for taking photos and one for delivering instructions. The number of the workers for machine learning tasks can be specified since it's meaningful to run parallel tasks in DPU.

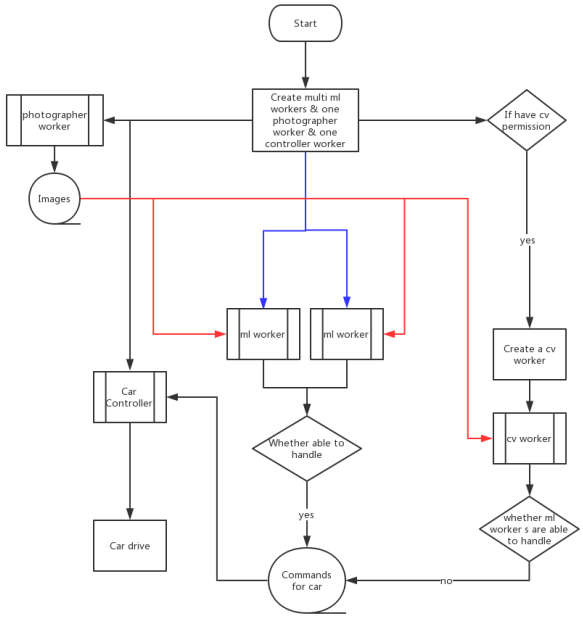

Fig. 3. Work Flow

C. AI model

The network we use is a simple end-to-end model. Since the DNNDK kit doesn't support some AI functions, we adapt

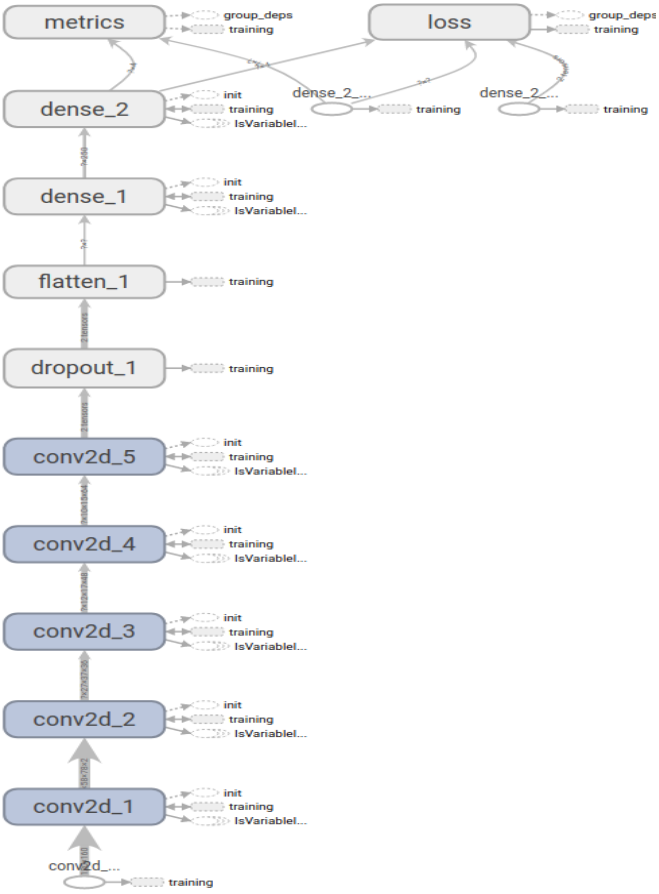

Fig. 4. Model Structure.

our model to DNNDK. The structure of the model is shown in Fig. 4.

The main part of the model is CNN layers, they can extract features from the images taken by the car's front camera. Some fully connected layers follow the CNN layers, they can finally extract the command information needed for auto-driving. The activation function we use is Relu because it works well and can be accelerated by DPU. The last layer is a Softmax layer, it can provide classification and normalization functions for the model.

Although the current model is not the perfect one and it cannot be accelerated by DPU totally, it's easy to make changes and optimizations to the model. With the development of DNNDK, more portion of the model can be accelerated and the whole system can be more efficient and powerful.

D. Simulator

Now we have built a simulator based on sdsandbox [13] to collect data for training, and we can also test our model using the simulator. With this tool, we can further separate software part from hardware part, even you don't have a real car, you can build and test your model in the virtual environment. The interface of our simulator is shown in Fig. 5.

The main part of the model is CNN layers, they can extract features from the images taken by the car's front camera. Some

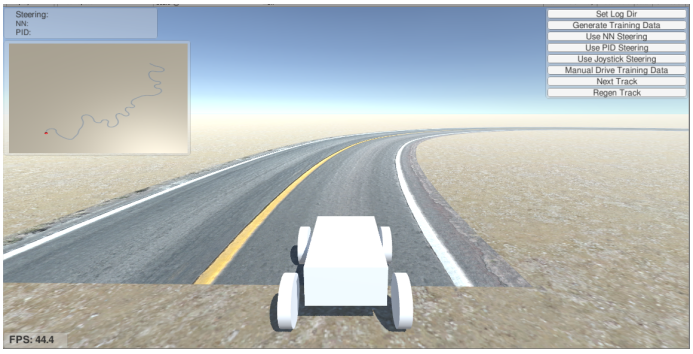

Fig. 5. Simulator Interface.

fully connected layers follow the CNN layers, they can finally extract the command information needed for auto-driving. The activation function we use is Relu because it works well and can be accelerated by DPU. The last layer is a Softmax layer, it can provide classification and normalization functions for the model.

Although the current model is not the perfect one and it cannot be accelerated by DPU totally, it's easy to make changes and optimizations to the model. With the development of DNNDK, more portion of the model can be accelerated and the whole system can be more efficient and powerful.

E. Algorithms

- Crossroad control.
  Since the AI model cannot handle the situation when the car meets a crossroad. We will help the car decide which direction to take according to the competition's request. The car will use neural network or cv algorithms to recognize all the scenes that need it to make a choice, for example the crossroad. After the car meets these scenes, first the main controller will tell the car which direction to take, then the car will use predefined control commands to drive through the crossroad. When this progress is finished, the AI model will retake the control of the car.
- Image process.
  Although our solution is end-to-end, the training data needs to be processed. Now we just do the normalization to the input images, later we will try some more according to the evaluation of the model's performance.
- Obstacle avoidance.
  For obstacle avoidance, traditional methods will first recognize the objects from the image and then use the state machine functions to make decisions. Our solution is quite different from it, we will add the obstacle avoidance scenes to the step of collecting training data. As a result, when the training step is finished, the AI model can avoid the obstacle automatically.
- Road lane detection.
  To detect road lane, we first use 5*5 Gaussian filter to remove the noise of the image. Then Canny edge detection [14] is applied to detect the edges of the image. Hough transform [15] is employed to detect the lines of

the image. Based on the orientation and relative position of detected lines, side line and stop line of the road can be identified.

To improve the robustness and accuracy of the algorithm, more techniques will be added to the algorithm, such as K-Means clustering [16] for Hough lines, Kalman/Gabor filtering for sampled data, alternative IPM (inverse perspective mapping) lane detection method [17].

Road lane detection algorithm is implemented in C++ with OpenCV. And it can be very time consuming running in ARM Cortex-A9 in Xilinx PYNQ-Z2. Some measures are taken to alleviate the heavy computational task of the algorithm. Firstly, the images captured by camera are cropped and down-sampled to reduce the computation complexity. Some time consuming tasks such as Gaussian filter, Canny edge detection, Hough transform are moved to FPGA using Xilinx xfopencv library [18].

- Traffic light recognition.

  To recognize the traffic light in the contest, we first use IHLS color space [19] to remove the disturbance of the color in complicated background. After obtaining the color information, Hough circle transform [20] is applied to get the shape information of the traffic light. Combing the shape and color information, traffic light or similar objects can be detected. Finally, a simple BP neural network is employed to accurately recognize the traffic light in the contest.

  Traffic light recognition algorithm is implemented in C++ with OpenCV. Similarly, it can be very time consuming running in ARM Cortex-A9 in Xilinx PYNQ-Z2. To reduce computation complexity, images captured by camera can be cropped as well since the traffic light only appear in the upper half of the image. Hough circle transform can be implemented in FPGA using Xilinx xfopencv library and BP neural network can be implemented in FPGA using Xilinx Vivado HLS.

IV. Development status

Until now, we have built the whole system, the AI model works well in simple scenes in virtual environment. We are busy implementing the cv functions and optimizing the hardware structure of the car. Later we will try to make some improvements to the model and do some tests and modifications according to the competition's rules.

V. Conclusion

The DPU embedded in FPGA accelerates the inference progress of AI algorithms. Our team wants to use this great power to build an autonomous driving system, with the help of Xilinx's Pynq-Z2 and DNNDK, our project will provide people with a new version of end-to-end autonomous driving solution.

We will focus on the stability of the system and the cooperations between AI network and traditional methods in the next step.

Acknowledgment

The project is supported by Xilinx and Institute of Computing Technology Chinese Academy Of Science.

References

[1] Nurvitadhi, E., Sheffield, D., Sim, J., Mishra, A., Venkatesh, G., & Marr, D. (2016, December). Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC. In 2016 International Conference on Field-Programmable Technology (FPT) (pp. 77-84). IEEE.

[2] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., ... & Boyle, R. (2017, June). In-datacenter performance analysis of a tensor processing unit. In 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA) (pp. 1-12). IEEE.

[3] Mahajan, Divya, et al. "Tabla: A unified template-based framework for accelerating statistical machine learning." 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2016.

[4] TUL PYNQ-Z2 [ONLINE] Available at: http://www.tul.com.tw/ProductsPYNQ-Z2.html

[5] TUL DPU [ONLINE] Available at: https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_0/pg338-dpu.pdf

[6] An FPGA design competition for developing autonomous driving robot cars is held in FPT2019 [ONLINE] Available at: http://fpt19.tju.edu.cn/Contest/FPT2019_FPGA_Design_Competition/Contents_and_Conditions.htm

[7] Kojima, A., & Nose, Y. (2018, December). Development of an Autonomous Driving Robot Car Using FPGA. In 2018 International Conference on Field-Programmable Technology (FPT) (pp. 411-414). IEEE.

[8] Aoto, M., Wada, Y., & Numata, Y. (2018, December). Development of an FPGA Controlled" Mini-Car" Toward Autonomous Driving. In 2018 International Conference on Field-Programmable Technology (FPT) (pp. 400-402). IEEE.

[9] Tawn Kramer. (2019, September) [ONLINE] Available at: https://github.com/tawnkramer/sdsandbox

[10] TUL Unity3d Available at: https://unity.com/learn

[11] TUL PetaLinux [ONLINE] Available at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug1144-petalinux-tools-reference-guide.pdf

[12] TUL DNNDK [ONLINE] Available at: https://www.xilinx.com/support/documentation/sw_manuals/ai_inference/v1_5

[13] Tawn Kramer, sdsandbox [ONLINE] Available at: https://github.com/tawnkramer/sdsandbox

[14] John Canny, "A Computational Approach to Edge Detection, "IEEE Transactions on Pattern Analysis and Machine Intelligence, pp.679-698, 1986.

[15] J. B. Burns, A. R. Hanson and E. M. Riseman, "Extracting Straight Lines," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 4, pp. 425-455, July 1986.

[16] S. Selim, M. Ismail, "K-Means-Type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optimality," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 6, no. 1, pp. 81-87, 1984.

[17] Jun Wang, Tao Mei, Bin Kong and Hu Wei, "An approach of lane detection based on Inverse Perspective Mapping," 17th International IEEE Conference on Intelligent Transportation Systems (ITSC), Qingdao, 2014, pp. 35-38.

[18] Xilinx. (2019, Sep) Xilinx/xfopencv. [Online]. Available: https://github.com/Xilinx/xfopencv

[19] Allan Hanbury and Jean Serra, "A 3d-polar coordinate colour representation well adapted to image analysis," in Proceedings of the Scandinavian Conference on Image Analysis (SCIA), 2003, pp. 804–811.

[20] M. Rizon et al., 2005, "Object Detection using Circular Hough Transform", American Journal of Applied Sciences 2, vol 12, pp. 1606-1609.