# Towards Executing Computer Vision Functionality on Programmable Network Devices

René Glebke
Chair of Communication and Distributed Systems
RWTH Aachen University
glebke@comsys.rwth-aachen.de

Johannes Krude
Chair of Communication and Distributed Systems
RWTH Aachen University
krude@comsys.rwth-aachen.de

Ike Kunze
Chair of Communication and Distributed Systems
RWTH Aachen University
kunze@comsys.rwth-aachen.de

Jan Rüth
Chair of Communication and Distributed Systems
RWTH Aachen University
rueth@comsys.rwth-aachen.de

Felix Senger
Chair of Communication and Distributed Systems
RWTH Aachen University
senger@comsys.rwth-aachen.de

Klaus Wehrle
Chair of Communication and Distributed Systems
RWTH Aachen University
wehrle@comsys.rwth-aachen.de

## ABSTRACT

By offering the possibility to already perform processing as packets traverse the network, programmable data planes open up new perspectives for applications suffering from strict latency and high bandwidth requirements. Real-time Computer Vision (CV), with its high data rates and often mission- and safety-critical roles in the control of autonomous vehicles and industrial machinery, could particularly benefit from executing parts of its logic within network elements.

In this paper, we thus explore what it takes to bring CV to the network. We present our work-in-progress efforts of implementing a line-following algorithm based on convolution filters on a P4-programmable NIC. We find that by appropriately identifying regions of interest in the image data and by diligently distributing the necessary calculations among the various match/action stages of the ingress- and egress pipelines of the NIC, our prototypical implementation can achieve over 19 decisions per second on 640x480 px grayscale images with filters large enough to guide a small autonomous car through various courses.

## CCS CONCEPTS

• **Networks** → **In-network processing**; *Middle boxes / network appliances*; *Programmable networks*; • **Computing methodologies** → Computer vision;

## KEYWORDS

In-network processing; P4; computer vision; convolution filters

**ACM Reference Format:**
René Glebke, Johannes Krude, Ike Kunze, Jan Rüth, Felix Senger, and Klaus Wehrle. 2019. Towards Executing Computer Vision Functionality on Programmable Network Devices. In *1st ACM CoNEXT Workshop on Emerging in-Network Computing Paradigms (ENCP '19), December 9, 2019, Orlando, FL, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3359993.3366646

*ENCP '19, December 9, 2019, Orlando, FL, USA*


## 1 INTRODUCTION

The introduction of Software-Defined Networking (SDN) and the associated programmability of data planes [5] has rekindled the hopes of many researchers to finally conceive solutions for scenarios with extremely tight latency constraints and high bandwidth requirements [24]. By partially offloading processing steps to network elements and thus avoiding overheads introduced by multi-hop paths and network stacks of operating systems, the applicability of In-Network Processing (INP) has already been shown to be beneficial for network management-related applications such as load balancing [20] or the detection of heavy hitters [29] and of DDoS attacks [16]. The possibility to reduce the overall traffic in map-reduce and machine learning applications via in-network data aggregation has been studied in [24, 25], as has the reduction of latency in industrial feedback control by offloading simple control algorithms to emulated programmable switches [23]. Further work concerns e.g., consensus protocols [18] and key-value caching [11].

A common pattern in most of the aforementioned works is that while the amount of data items handled in total is large (up to the saturation point of the switching fabric in the network management examples), the size of the individual items processed is small (fitting into single packets) and the processing executed on each item is not very complex (a bit of housekeeping few minor arithmetic operations). In essence, each decision here can be taken on the data plane at line rate, i.e., via a single pass over a single packet. As we will describe later in this paper, the paradigms in popular data plane programming languages such as P4 [4] are especially amenable to such settings.

Prime examples for scenarios where the aforementioned assumptions on data sizes and decision complexities do not hold are processes aided by Computer Vision (CV) functions such as edge detection and object recognition in images. Involved in the control of industrial processes [13, 32] or in finding trajectories of autonomous cars [8] or swarms of drones [33], CV methods nowadays help operating a vast variety of systems. In some cases, e.g., when used to detect pedestrians in autonomous driving, they even provide

vital safety information. Executing parts of the CV logic within the network (e.g., as inputs to industrial control systems) could further reduce reaction times and hence improve the general behavior and safety of such systems. However, even low-resolution camera images are at least two orders of magnitude larger than ordinary Ethernet packets [7], so that naïve approaches may require keeping large states over time spans much longer than the ordinary dwelling times of packets within network elements, resulting in potentially prohibitive memory requirements for in-network devices. Furthermore, CV algorithms typically involve computationally complex operations such as matrix multiplications (on these large amounts of data), which may not be possible on current Programmable Network Devices (PNDs) while sustaining line rate.

**Contributions and Structure.** In this paper, we explore to which extent the processing capabilities of programmable data planes lend themselves to the high demands of CV algorithms. To this end, we first give a short introduction into the data processing pipelines of current PNDs using the example of $P4_{16}$ [4, 31] programs (Section 2). Next, we devise a simple line-following scenario as an example CV application, discuss possibilities to offload parts of the necessary computations to the network, and present details of our prototypical INP pipeline (Section 3). Afterwards, we evaluate the applicability and scalability of our INP-based line-following approach using both synthetic benchmarks and a real-world testbed with a small autonomous car (Section 4), before briefly reviewing related work in Section 5 and concluding our paper in Section 6.

## 2 BACKGROUND: THE P4 PIPELINE FOR PROGRAMMABLE DATA PLANES

While the idea of processing data as packets advance through networks has been repeatedly proposed in past decades [28, 30, 34], only the fairly recent availability of PNDs (e.g., [3, 21]) called for a joint effort by major players from industry and research to introduce a common and open programming language for the data plane, termed *P4* [4]. The language introduces an abstract forwarding model for PNDs based on the notion of Match-Action Units (MAUs), which allows specialized hardware to process packets at a fixed latency with a throughput of one packet per clock cycle [5].

An arriving packet is first *parsed* and checked against a number of anticipated headers (e.g., TCP/IP) and enriched with *metadata*. The parsed headers are then passed to a sequence of MAUs, which first *match* the content of one or several header fields against expected values or ranges, often using table lookups. Upon finding a match, the PND takes a corresponding *action*, such as manipulating the content of the packet's headers with simple arithmetic operations or marking the packet for redirection to a specific output port. Several MAUs can be stitched together to form a processing *pipeline*, and both per-packet metadata as well as metadata shared between packets (in designated *registers* of the device) can be used to pass information from one MAU to the next. P4 conceptually distinguishes between the *ingress* part of the pipeline, which can process data before the next hop of a packet has been determined, and an *egress* part for checks and manipulations before the packet is sent out again.

The expressiveness of actions in P4 depends on the target hardware the program is executed on, and the language mandates only a small number of operations [31]. It, e.g., neither defines floating-point operations, nor the amount of individual operations an action must at least contain and not even the number of available MAU stages. More complex processing may hence need to be reformulated according to the available functionality [23]. Some targets may offer the possibility to re-inject packets at the beginning of the pipeline (*recirculation*), allowing it to pass through the same MAUs multiple times, which enables the implementation of longer processing procedures. A packet traversing the pipeline once can sustain line rate [17]. Recirculation in an ingress-egress-ingress-... fashion in contrast causes packets to stay in the PND longer and may potentially cause side effects such as buffer overflows on the ingress side of the pipeline if the rate of incoming plus recirculated packets surpasses the data plane's capacity, yet it may be the only possible method to implement longer processing procedures.

Thus, while P4's pipeline is a good fit for decisions that can be taken based on a single packet (e.g., next-hop lookups for routing), more complex decisions or such that need a sharing of information between several packets are, although desirable, much harder to properly express. In the following, we give an example of a common CV operation that, when executed within the network, suffers from both of the above problems, and is thus an ideal subject of study when aiming to compensate for the intricacies of P4's data plane programming approach.

## 3 A NETWORK-ASSISTED LINE-FOLLOWING SCENARIO

Computer vision is an interdisciplinary field comprising a variety of tasks which can be summarized as the "recognition, reconstruction and reorganization" [19] of data to help machines gain higher-level understandings of images and image sequences. Depending on the task at hand, images are processed using highly individual processing sequences of varying complexity (see, e.g., [22, 26, 35]). In this section, we construct an example sequence that allows us to recognize the middle of a line highly contrasting with the background, in order to guide a small autonomous car along that line.

### 3.1 Scenario Overview

In our scenario, we search a specific horizontal region of images taken by a camera mounted on a small car for a pair of edges that are most distinguishable from a potentially noisy background (an example image from our setup can be seen in Figure 1). We then assume that the edges we find while searching within that region bound a broad vertical line we intend to follow with the car. To this end, we define both a left and a right threshold (also visible in Figure 1). If the middle of the found edges (and thus the middle of the assumed line) deviates so much from the middle of the image that it passes one of these thresholds, we turn our car into the direction of the deviation so that a subsequent forward movement advances the car along the line again. While there exist more sophisticated line-following methods (e.g., using multiple scanning regions at different vertical offsets to break ties when a horizontal line crosses one region), our simple approach allows us to concentrate our efforts on the actual CV processing part rather than on non-CV-related functionality. We hence only explore the possibility of implementing the edge detection functionality

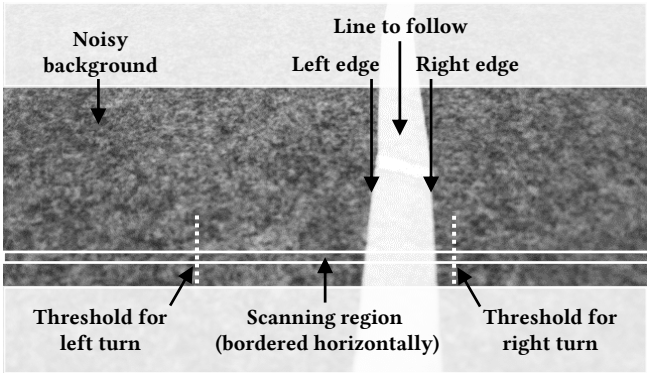


**Figure 1: Example image from our line-following scenario. A PND scans a horizontal region of an image from a camera on a small car for the edges of a sharply contrasting vertical line. The edges' positions are sent back to the car, which turns accordingly if the positions surpass a threshold for either side. It then advances slightly forward and sends the next picture.**

with the help of PNDs and implement the rest of the functionality (e.g., controlling the car based on the found edge positions) in a *client* application on the platform hosting the camera. In the following, we describe an approach to the edge detection problem which is popular in autonomous driving scenarios [22] and exhibits properties very amenable to execution within PNDs.

## 3.2 Edge Detection via Convolution

While there are plenty of edge detection algorithms available, a simple yet useful technique involves the application of two *filter matrices* to an image: By convolving an image $P$ pixel-by-pixel with a specific $m \times n$ *filter matrix* $F$ and storing the values of the convolution operations in a result matrix $R$, i.e.,

$$R(x,y) = \sum_{i=1}^{m} \sum_{j=1}^{n} P(x-i+a, y-j+a)\, F(i,j)$$

(where $a$ denotes the middle coordinate of $F$, and $P(x,y) = 0$ for coordinates outside the image), we gain entries of $R$ that – in a geometric interpretation – represent the dot product and hence the similarity between the filter and the corresponding region of the image. In grayscale images, we can, e.g., detect edges using the so-called *Prewitt operator* [2], which chooses two filter matrices $F_{\Delta_H}$ and $F_{\Delta_V}$ of dimension $3 \times 3$, so that the entries in $R$ correspond to the similarity of the respective region of an image with a sharp horizontal ($F_{\Delta_H}$) or vertical ($F_{\Delta_V}$) edge; the higher the value of an entry, the higher the resemblance of the region with an edge in the direction of the passing of the filter. Thus, the left edge of a highly-contrasting vertical line as in our example scenario can be found by keeping track of the highest response of convolutions within our scanning region with $F_{\Delta_V}$, and the right edge using a convolution with $-F_{\Delta_V}$. The standard variant of the Prewitt operator only takes the 1-pixel neighborhood into account to find edges, but it can be enlarged to cover the further vicinity, and we use a method similar to the one described in [27] to construct larger filters to better account for the noisy background in our scenario.

**Applicability on PNDs.** The Prewitt operator and related convolution filters have three features that make them especially interesting for partially offloading them to the network. First, the calculation of the discrete convolution only needs addition, subtraction and multiplication operations of integer data (pixel intensity values), and these are mandatory for all P4-enabled devices to implement [31]. Second, the individual entries of the result matrix are only correlated to a small part of an image. Thus, a PND implementing a convolution-based CV stage does not need to wait until a full image has been received to begin processing. Third, all entries of the result matrix can be calculated independently. This means that once all convolutions involving one row (or column) of pixels have been calculated, this row (column) can be discarded by the device, hence reducing the amount of state the device needs to hold in memory. Convolution filters thus are highly versatile tools in the CV domain implementable with both low arithmetic requirements and a low memory footprint, making them promising candidates for implementation via INP principles. We now discuss how we can specifically offload our filter-based edge detection step to the network in our example setup.

## 3.3 Bringing a Convolution-based Edge Detection Filter to the Network

Having found a suitable candidate for edge detection on PNDs, we now implement our line-following scenario according to the structure presented in Section 3.1. Our client controlling the car is implemented in Python and communicates with the PND via standard Linux sockets, while our filter is implemented on a PND using $P4_{16}$. We present details of our prototypical implementation in the following.

**Communication pattern.** For each movement decision we want to take, our client sends image data to the PND. To this end, it splits a (grayscale) camera picture into a series of chunks of $n \times n$ pixels, as depicted in part Ⓐ on the left of Figure 2. It then prefixes each chunk with the ID of the picture and the position of the chunk within the picture (*chunk ID*), and then sends each prefixed chunk to the PND in a separate UDP/IP packet in ascending order of chunk ID. While this approach limits the achievable throughput due to the overhead introduced by the individual packaging, it allows us to both drastically simplify the implementation of the edge detection filter in our prototype (as further described below) and to test our approach with data streams of many small items which all require individual handling by the PND. Upon packet arrival, the PND uses P4's ingress parser to separate picture- and chunk IDs from the image data, and subsequently enters the beginning of the actual convolution procedure (parts Ⓑ through Ⓔ of Figure 2). After the convolution process has completed for the entire scanning region, we send out a packet containing the found left and right edges of the line to our client. Since we cannot create packets in P4 [31], we rewrite the packet containing the last chunk of the scanning region to contain our edge positions and send it back to the client.

**Reduction of computation efforts.** Due to the limited capacity of PND pipelines, we first try to reduce the computation efforts for the edge detection to a minimum. In our scenario, it is sufficient

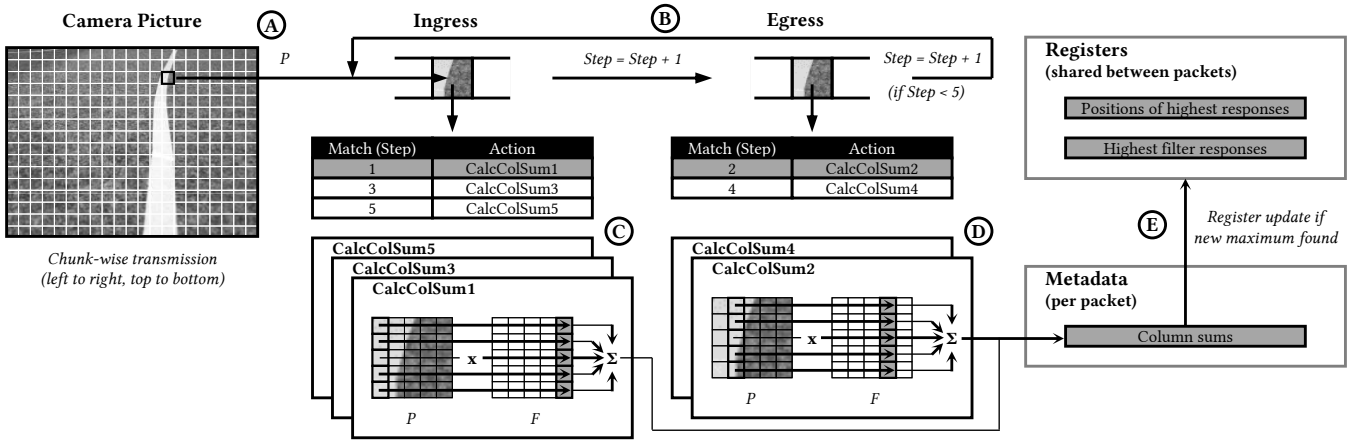


**Figure 2: Simplified overview of our edge detection approach on PNDs (from left to right): Ⓐ A client sends a chunked camera picture $P$ (here: $5 \times 5$ pixels) to a PND. Ⓑ The PND uses P4 match-action stages in ingress Ⓒ and egress Ⓓ to apply an edge detection filter $F$ *column-wise*. Ⓔ Registers hold information between packets.**

to calculate convolutions for our scanning region. However, our client is unaware of the exact vertical position of this region within the picture and hence sends the entire picture to the PND. Thus, to reserve as much of the PND's MAU- and buffer capacity as possible, we immediately drop any packet not containing chunks within the scanning region. Further, we assume that our scanning region is contained within exactly one horizontal row of chunks sent by the client, i.e., we limit the size of our filters so that they only consider pixels within that series of chunks. Otherwise, depending on the size of our filter matrices (cf. Section 3.2), we may need to keep information from several rows of chunks in the PND's memory to perform our convolution calculations, which would be hard given the limited memory of PNDs. To further reduce our computation effort, we assume that the vertical line we want to detect has a very high contrast to the background, similar to the ones visible in Figure 1 and on the left side of Figure 2. This allows us to assume that the line is well detectable on any horizontal part of the picture, and, especially, on any of the rows that constitute our scanning region. We thus choose to calculate the convolution around the pixels in the *middle row* of the scanning region only, since this should yield sufficient results for our prototypical implementation. At last, since the filter matrices for both edges are negative versions of each other (again, cf. Section 3.2), we can calculate the partial sums for the two convolutions in a unified manner, using the negative version of the left edge's summand also for the right edge to further reduce the number of calculations to perform.

**P4 Pipeline Organization.** While we proactively drop unneeded packets and reuse as many results of arithmetic operations as possible, depending on the architecture of the PND, a single pass through the pipeline may not be sufficient to complete a convolution. To make the most use of the P4 pipeline on PNDs, we implement our edge detection as a *looped program* that executes calculations for the convolution in both the ingress and the egress stages and then recirculates the packet if more calculations are needed (part Ⓑ in the upper middle of Figure 2). We use a per-packet metadata field to store the current processing *step* and employ two tables that match on this field (one in ingress, one in egress) to trigger the calculations in the corresponding actions. Our calculation steps are organized along the horizontal axis, i.e., we first compute the partial sums for the leftmost column of our picture chunk in the ingress (part Ⓒ in the lower middle of Figure 2), then the sums for the second column in the egress (part Ⓓ), and so forth. The sums for the columns are stored in per-packet metadata variables (part Ⓔ), so we can calculate the final values for the filters (the *responses*) once all column sums are available. We keep track of the highest responses of our two filters over all considered chunks in registers (also in part Ⓔ) and ultimately return the positions that yielded these responses as the positions of the edges.

## 4 EVALUATION

We present the results of a preliminary evaluation of our implementation of a line-following filter on PNDs. To this end, we compile our P4 program with a Netronome Agilio CX 2x25GbE SmartNIC [21] as the target, using the Netronome-specific P4 compiler with default settings. We use a second SmartNIC of the same type without a P4 program installed to send images to our program, both real images from our test setup in Section 3.1 (with the car connected wirelessly to the computer of the second SmartNIC) and synthetic variants from an image generator. Note that the results of the performed convolutions and hence the found edge positions are independent of the platform they are calculated on as long as integer calculations of sufficient bit length are available (our filters require signed 32-bit integers). We thus only evaluate the scalability of our approach regarding maximum filter size and the time our pipeline requires to calculate the positions of the edges, yielding the number of pictures we can analyze per second.

**Maximum filter size.** We first attempt to find the maximum size of a P4 program implementing our looped filter by synthesizing programs with increasing number of columns and rows per chunk (and, hence, allowed filter size). For a filter size of $m \times n$, each program requires $n$ 32-bit metadata fields to store the column-wise

convolution summands, 2 32-bit registers to store the overall highest responses, as well as 4 16-bit registers to store control information. Additionally, we need 13 bit of further control-flow related metadata per packet, including 8 bits to store the current step in the recirculation pipeline. We are able install P4 programs with a maximum filter size of $10 \times 10$ pixels on the SmartNIC; programs with larger filter sizes do seem to compile as valid P4 code, but the tool chain is unable to synthesize the byte code for the device. The number of match-action stages in our looped variant remains constant irrespective of the filter size, and we assume that the number of calculations we execute per stage surpasses the capacity of the SmartNIC's MAUs implementation at this size. Filter sizes larger than of $10 \times 10$ pixels may appear in more elaborate object recognition scenarios [14]. However, based on a visual inspection of the accuracy of the edge finding process (by rendering the position of the found edges into the respective photos taken by the camera) and the resulting behavior exhibited by our car, we find that already when applying a filter of size $5 \times 5$ pixels, we can guide our car through a course with high background noise and various sharp bends most reliably in terms of accuracy of the detected edge positions.

**Time per picture / throughput.** To assess the number of pictures we can process per second using our approach, we use our image generator on the machine hosting the second SmartNIC to send out images to our P4 program as quickly as possible using a Linux socket with default settings, and measure the processing time exhibited by our program, averaged over 1000 pictures. We capture the time from the beginning of the transmission of the first chunk of a picture until the arrival of the packet containing the positions of the edges, and additionally use the SmartNIC's timestamping functionality to assess the time it takes for a single chunk to circulate through our pipeline. Using the $5 \times 5$ filter, due to the large overhead introduced by sending packets with a payload of essentially 25 bytes (the size of the filter), we can send and analyze one picture approx. every 50 ms (stddev 3 ms). Processing the last chunk for each picture takes approx. 150 $\mu s$ (stddev 1.3 $ms$). The SmartNIC however drops packets intermittently for this filter size; we do not get an answer for 13.7% of our pictures. We attribute this behavior to buffering problems caused by recirculating a large number of packets several times. For the $10 \times 10$ filter with 800 bytes per chunk, we experience zero drops and the achievable rate increases to approx. one picture every 13 ms (stddev 5 ms), while the processing time for the respective last chunk increases to approx. 187 $\mu s$ (stddev 0.6 $ms$). These results mean that we can perform edge detection in the network on up to 19 images per second using the reliable $5 \times 5$ pixels filter, and on up to 77 images using the less precise $10 \times 10$ pixels filter, allowing the car to move at a considerable speed. Yet, we still have to find the optimal trade-off between adequate filter size and a steady achievable throughput in our prototype.

## 5 RELATED WORK

The execution of CV functionality has been studied for a number of restricted platforms. Wang et al. [33], e.g., use the example of flying drones and apply a number of pre-processing steps on the drones to reduce bandwidth usage when sending video streams to edge computing services, whereas Denby et al. [6] adapt CV to the resources of low-cost nanosatellites. MobileNets [9] and SqueezeNet [10] propose partitioning- and scaling methods for convolutional neural networks to accommodate CV functionality on devices with limited processing power, memory, or connection bandwidth. All these approaches consider restricted CPU/GPU-based platforms, while other works, such as the ones by Korol et al. [12] and Aguilar-González et al. [1], target ASICs and FPGAs. To the best of our knowledge, we are the first to show that a simple CV task can also be performed on the match-action pipelines of modern PNDs, and thus, as a *service* offered by the network, allowing restricted platforms of varying kinds to benefit.

## 6 CONCLUSION

In this paper, we describe our work-in-progress efforts to execute Computer Vision (CV) functionality on Programmable Network Devices (PNDs). We identify convolution-based filters (which are common steps in many CV scenarios) as promising candidates for an execution on PNDs and implement an edge detection filter on a PND using the P4 programming language. This allows us to guide a small autonomous car over a variety of courses with up to 19 decisions per second using a reliable $5 \times 5$ pixels filter. In our implementation, we use the nature of P4's processing pipeline in combination with the packet-oriented working mechanism of networks to our advantage, by diligently splitting up the processing load, distributing it along the pipeline and sharing information obtained from the processing of a single previous packet with the packet currently in the pipeline.

Our implementation is prototypical and can be improved on in a variety of ways. We may, e.g., be able to support even larger filter sizes by splitting up the per-column calculations into additional sub-stages, alleviating the problem of overfull action stages, which currently appear as a limiting factor. To this end, we also want to explore how we can make more use of the available table memory on the PNDs. Our calculations for the convolution filters are currently explicit. It would thus be interesting to see whether data sent to the devices can be structured in such a way that calculations can be assisted via table lookups, e.g., to mitigate the problem of intermittent drops with small filter sizes. We also made a number of assumptions concerning the structure and the arrival pattern of picture data at the PNDs. Most significantly, our approach is currently only able to calculate the convolution filter on a per-packet basis, i.e., we sub-sample the horizontal region we search in. Although our experimental evaluation shows that this is sufficient for line-following scenario, we cannot deal with cases where an edge is split up into multiple packets. We thus plan to investigate whether it is possible to share data between packets that are processed in the pipeline such that the convolutions can also be calculated across packet borders. Furthermore, given its usage of recirculation and potentially sparse global and metadata memory, we plan to evaluate the impact of our filter on co-located [15] in-network functions.

Our initial results are promising and we believe that further CV functionality and, more generally, other *stream-based* data such as sensor values can be handled within the network using processing pipelines similar to ours.

## ACKNOWLEDGMENTS

The authors would like to thank the German Research Foundation (DFG) for the kind support within the REFLEXES project (ID 315171171) in Priority Programme 1914 "Cyber-Physical Networking", the Collaborative Research Centre 1053 "MAKI — Multi-Mechanisms-Adaptation for the Future Internet" (ID 210487104), and the Cluster of Excellence "Internet of Production" (ID 390621612).

## REFERENCES

[1] Abiel Aguilar-González, Miguel Arias-Estrada, Madaín Pérez-Patricio, and J. L. Camas-Anzueto. 2019. An FPGA 2D-convolution unit based on the CAPH language. *Journal of Real-Time Image Processing* 16, 2 (01 Apr 2019), 305–319. https://doi.org/10.1007/s11554-015-0535-1

[2] I. Ahmad, I. Moon, and S. J. Shin. 2018. Color-to-Grayscale Algorithms effect on Edge Detection – A Comparative Study. In *Proceedings of the 2018 International Conference on Electronics, Information, and Communication (ICEIC) (ICEIC '18)*. IEEE, New York, NY, USA, 1–4. https://doi.org/10.23919/ELINFOCOM.2018.8330719

[3] Barefoot Networks. 2019. The World's Fastest & Most Programmable Networks. (2019). https://www.barefootnetworks.com/resources/worlds-fastest-most-programmable-networks/

[4] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *ACM SIGCOMM CCR* 44, 3 (2014), 87–95.

[5] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN. In *Proceedings of the 2013 ACM SIGCOMM Conference (SIGCOMM '13)*. ACM, New York, NY, USA, 99–110. https://doi.org/10.1145/2486001.2486011

[6] Bradley Denby and Brandon Lucia. 2019. Orbital Edge Computing: Machine Inference in Space. *IEEE Computer Architecture Letters* 18, 1 (2019), 59–62. https://doi.org/10.1109/LCA.2019.2907539

[7] René Glebke, Martin Henze, Klaus Wehrle, Philipp Niemietz, Daniel Trauth, Patrick Mattfeld, and Thomas Bergs. 2019. A Case for Integrated Data Processing in Large-Scale Cyber-Physical Systems. In *Proceedings of the 52nd Hawaii International Conference on System Sciences (HICSS) (HICSS 52)*. University of Hawai'i at Manoa / AIS, Honolulu, HI, USA, 7252–7261.

[8] L. Heng, B. Choi, Z. Cui, M. Geppert, S. Hu, B. Kuan, P. Liu, R. Nguyen, Y. C. Yeo, A. Geiger, G. H. Lee, M. Pollefeys, and T. Sattler. 2019. Project AutoVision: Localization and 3D Scene Perception for an Autonomous Vehicle with a Multi-Camera System. In *Proceedings of the 2019 International Conference on Robotics and Automation (ICRA) (ICRA '19)*. IEEE, New York, NY, USA, 4695–4702. https://doi.org/10.1109/ICRA.2019.8793949

[9] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. (2017). arXiv:arXiv:1704.04861

[10] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. (2016). arXiv:arXiv:1602.07360

[11] Xin Jin, Xiaozhou li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. 2017. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM, New York, NY, USA, 121–136. https://doi.org/10.1145/3132747.3132764

[12] Guilherme Korol and Fernando Gehm Moraes. 2019. A FPGA Parameterizable Multi-layer Architecture for CNNs. In *Proceedings of the 32Nd Symposium on Integrated Circuits and Systems Design (SBCCI '19)*. ACM, New York, NY, USA, Article 30, 6 pages. https://doi.org/10.1145/3338852.3339840

[13] K. Kottari and V. Delibasis, K.and Plagianakos. 2018. Real time vision-based measurements for quality control of industrial rods on a moving conveyor. *Multimedia Tools and Applications* 77, 8 (01 Apr 2018), 9307–9324.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., Red Hook, NY, USA, 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[15] Johannes Krude, Jaco Hofmann, Matthias Eichholz, Klaus Wehrle, Andreas Koch, and Mira Mezini. 2019. Online Reprogrammable Multi Tenant Switches. In *1st ACM CoNEXT Workshop on Emerging in-Network Computing Paradigms (ENCP '19)*. ACM. https://doi.org/10.1145/3359993.3366643

[16] Â. C. Lapolli, J. Adilson Marques, and L. P. Gaspary. 2019. Offloading Real-time DDoS Attack Detection to Programmable Data Planes. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, New York, NY, USA, 19–27.

[17] Alberto Lernen, Rana Hussein, and Philippe Cudre-Maurox. 2019. The Case for Network-Accelerated Query Processing. In *9th Biennial Conference on Innovative Data Systems Research (CIDR '19)*.

[18] Jialin Li, Ellis Michael, Naveen Kr. Sharma, Adriana Szekeres, and Dan R. K. Ports. 2016. Just Say NO to Paxos Overhead: Replacing Consensus with Network Ordering. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*. USENIX Association, 467–483.

[19] Jitendra Malik, Pablo Arbeláez, João Carreira, Katerina Fragkiadaki, Ross Girshick, Georgia Gkioxari, Saurabh Gupta, Bharath Hariharan, Abhishek Kar, and Shubham Tulsiani. 2016. The three R's of computer vision: Recognition, reconstruction and reorganization. *Pattern Recognition Letters* 72 (2016), 4–14. https://doi.org/10.1016/j.patrec.2016.01.019 Special Issue on ICPR 2014 Awarded Papers.

[20] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *Proceedings of the 2017 ACM SIGCOMM Conference (SIGCOMM '17)*. ACM, New York, NY, USA, 15–28. https://doi.org/10.1145/3098822.3098824

[21] Netronome. 2019. Agilio CX SmartNICs. (2019). https://www.netronome.com/products/agilio-cx/

[22] Thiago Rateke, Karla A. Justen, Vito F. Chiarella, Antonio C. Sobieranski, Eros Comunello, and Aldo Von Wangenheim. 2019. Passive Vision Region-Based Road Detection: A Literature Review. *ACM Comput. Surv.* 52, 2, Article 31 (March 2019), 34 pages. https://doi.org/10.1145/3311951

[23] Jan Rüth, René Glebke, Klaus Wehrle, Vedad Causevic, and Sandra Hirche. 2018. Towards In-Network Industrial Feedback Control. In *Proceedings of the ACM SIGCOMM 2018 Morning Workshop on In-Network Computing (NetCompute '18)*. ACM, New York, NY, USA, 14–19. https://doi.org/10.1145/3229591.3229592

[24] Amedeo Sapio, Ibrahim Abdelaziz, Abdulla Aldilaijan, Marco Canini, and Panos Kalnis. 2017. In-Network Computation is a Dumb Idea Whose Time Has Come. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks (HotNets-XVI)*. ACM, New York, NY, USA, 150–156. https://doi.org/10.1145/3152434.3152461

[25] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan RK Ports, and Peter Richtárik. 2019. *Scaling Distributed Machine Learning with In-Network Aggregation*. Technical Report. KAUST. https://arxiv.org/abs/1903.06701v1

[26] Muhamad Risqi U. Saputra, Andrew Markham, and Niki Trigoni. 2018. Visual SLAM and Structure from Motion in Dynamic Environments: A Survey. *ACM Comput. Surv.* 51, 2, Article 37 (Feb. 2018), 36 pages. https://doi.org/10.1145/3177853

[27] J. Scharcanski and A. N. Venetsanopoulos. 1997. Edge Detection of Color Images Using Directional Operators. *IEEE Transactions on Circuits and Systems for Video Technology* 7, 2 (April 1997), 397–401. https://doi.org/10.1109/76.564116

[28] Beverly Schwartz, Alden W. Jackson, W. Timothy Strayer, Wenyi Zhou, R. Dennis Rockwell, and Craig Partridge. 2000. Smart Packets: Applying Active Networks to Network Management. *ACM Trans. Comput. Syst.* 18, 1 (Feb. 2000), 67–88. https://doi.org/10.1145/332799.332893

[29] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford. 2017. Heavy-Hitter Detection Entirely in the Data Plane. In *Proceedings of the Symposium on SDN Research (SOSR '17)*. ACM, New York, NY, USA, 164–176. https://doi.org/10.1145/3050220.3063772

[30] David L. Tennenhouse and David J. Wetherall. 1996. Towards an Active Network Architecture. *SIGCOMM Comput. Commun. Rev.* 26, 2 (April 1996), 5–17. https://doi.org/10.1145/231699.231701

[31] The P4 Language Consortium. 2018. $P4_{16}$ Language Specification version 1.1.0. (30 11 2018). https://p4.org/p4-spec/docs/P4-16-v1.1.0-spec.html

[32] M. Vogelbacher, J. Matthes, H. Keller, and P. Waibel. 2019. Progression and Evaluation of a Camera-Based Measurement System for Multifuel Burners under Industrial Process Conditions. *IEEE Transactions on Industrial Informatics* 15, 10 (Oct 2019), 5466–5474. https://doi.org/10.1109/TII.2019.2899946

[33] J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai, S. Yang, and M. Satyanarayanan. 2018. Bandwidth-Efficient Live Video Analytics for Drones Via Edge Computing. In *Proceedings of the 2018 IEEE/ACM Symposium on Edge Computing (SEC) (SEC '18)*. IEEE, New York, NY, USA, 159–173. https://doi.org/10.1109/SEC.2018.00019

[34] J. Zander and R. Forchheimer. 1988. The SOFTNET Project: A Retrospect. In *8th European Conference on Electrotechnics, Conference Proceedings on Area Communication (EUROCON '88)*. IEEE, New York, NY, USA, 343–345.

[35] Xin Zhang, Yee-Hong Yang, Zhiguang Han, Hui Wang, and Chao Gao. 2013. Object Class Detection: A Survey. *ACM Comput. Surv.* 46, 1, Article 10 (July 2013), 53 pages. https://doi.org/10.1145/2522968.2522978