Discrete event dataflow as a formal approach to specification of industrial vision systems*

Oleksandr Semeniuta[1] and Petter Falkman[2], *Member, IEEE*

***Abstract*— The need for more flexible manufacturing systems stimulates the adoption of industrial robots in combination with intelligent computing resources and sophisticated sensing technologies. In this context, industrial vision systems play a role of inherently flexible sensing means that can be used for a variety of tasks within automated inspection, process control and robot guidance. When vision sensing is used within a large complex system, it is of particular importance to handle the complexity by introducing the appropriate formal methods. This paper overviews the challenges arising during design, implementation and application of industrial vision systems, and proposes an approach, dubbed Discrete Event Dataflow (DEDF), allowing to formally specify vision dataflow in the context of larger systems.**

I. Introduction

Machine vision is a technological field applying computer vision methods for industrial needs. The latter include automatic inspection, process control, and robot guidance [1], [2], [3], [4]. Compared to other types of sensors, vision systems are highly reconfigurable and able to measure a wide range of characteristics. Despite the envisioned opportunities, vision systems in the industrial environment possess numerous challenges such as a number of factors influencing the algorithms, quality of system calibration, complexity of system development and high processing time.

Any vision system starts its work by acquiring an image or a set of images from cameras or data storage devices. After the original images are loaded into computer memory, a vision system exerts a certain set of operations upon them in order to obtain the final application-dependent information. The operations typically constitute the well-known image processing or computer vision algorithms, and their sequence resembles a pipeline, starting at the image acquisition phase and ending with obtaining the desired result. As an example, Figure 1 shows a sequence of operations that are common to many vision systems applications. In such general system, a camera acquires an image, which is then enhanced to simplify the later processing steps. After that, certain parts of the image are segmented, and the obtained parts are further used to detect the desired features.

Similar pipelines as described in Figure 1 are presented in numerous literature sources (including [2], [3], [4]) in order to visualize the principles of the particular vision

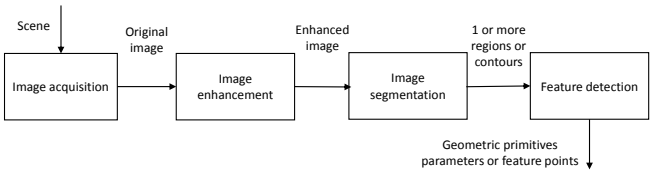

Fig. 1. Simple vision pipeline

systems. These pipelines may be more or less abstract, and may include additional graphical elements, such as flowchart conditionals, parameter lists etc. In any case, the major reason for depicting the pipelines lies in providing a better understanding of the respective vision system concepts. Despite being a proper visual aid, the traditional pipeline diagrams lack the possibility of applying formal modeling, specification and analysis methods.

Formal methods, originated in the areas of computer science and systems engineering, are based on the idea that a system can be described as a formal model, which can be used for the purposes of system optimization, verification and synthesis. The need for using formal methods often arises in complex systems design, where the correct behavior is critical. Additionally, the unambiguity of formal representation allows for ensuring common understanding and fostering the creation of the appropriate algorithmic methods. In manufacturing, formal methods are used for system modeling, simulation, control, performance evaluation and fault diagnosis [5].

This paper presents a concept of Discrete Event Dataflow (DEDF), a dynamic dataflow formalisms enhanced by discrete event semantics. The goal of DEDF is to provide a formal apparatus for specification of vision systems in the context of larger systems utilizing computer vision as one of their sensing means. The proposed specification language should (1) provide an unambiguous understanding of the working principle of the modeled vision system, (2) be technology-agnostic, i.e. not tailored to specific hardware or software, and (3) provide the ability to integrate vision models with the models of the larger controlled system.

This paper is organized as follows. Section II overviews the background literature on the topics of industrial application of vision and the existing approaches of formal modeling of vision systems. Section III presents the proposed formalism of Discrete Event Dataflow (DEDF) and formal notation for common vision data types. Section IV applies DEDF to modeling of a camera calibration system and passive stereo vision system. Section V provides a discussion

*This work was supported by the Norwegian Research Council

[1]Oleksandr Semeniuta is with Faculty of Technology, Economy and Management, Gjøvik University College, 2815 Gjøvik, Norway oleksandr.semeniuta@hig.no

[2]Petter Falkman is with Department of Signals and Systems, Chalmers University of Technology, Göteborg, Sweden petter.falkman@chalmers.se

about the presented method, including its limitations and perspectives for future development.

II. Background literature

A. Industrial vision

In the industrial context, computer vision is often used as a sensing means in control systems that provide actuation function on the mechanical components. The processes in which machine vision takes part include automatic inspection, process control, and robot guidance. Vision in the context of automatic inspection is particularly useful in cases when the parts are inaccessible otherwise, when a large number of small features has to be measured, or when the parts have complex shapes [6]. In robot control, vision systems provide flexibility of robot operation and reduce the necessity of fixed-point robot motion programming.

The inspected features of industrial product or process that are measured by vision systems include dimensional quality, structural quality, surface quality, and operational quality [2].

The use of vision systems is a non-trivial field, associated with a number of problems and challenges. There exist numerous factors that influence the accuracy of vision algorithms, including the measured object characteristics (size, shape, color, texture), camera characteristics (camera resolution, quality of lenses), environment characteristics (pose, illumination) [2], [7]. In addition, the way the vision algorithms is developed plays a significant role for the quality of measurement.

Since machine vision is used for controlling mechanical equipment such as robots, real-world measurements are often required. Camera calibration, extrinsic calibration and stereo vision system calibration processes provide the necessary parameters and rigid transformations making possible to operate in real-world space. The quality of the applied calibration methods directly impacts the precision of robot operations, and therefore have to be acceptable.

Other challenges include expensive software development and high processing time [7].

B. Dataflow and actor model

One way to model cyber-physical systems is by applying an actor-oriented design approach [8], [9]. Actors constitute distinct concurrently-run computational components that communicate through signal ports. The interaction rules between actors are defined by the respective *models of computations (MoCs)*, which include discrete events, finite state machines, continuous time, synchronous reactive, and dataflow [9].

The dataflow model of computation is particularly suitable for *streaming applications*, where signals are routed through computations [10], [11]. As presented in figure 2, in the dataflow model, a concrete signal produced by the output port of actor A_i and consumed by the input port of actor A_j is modeled as a token. The firing rule of an actor specifies how many tokens are required on the input port in order for the actor to *fire*. If for actor A_j this number is M_j, then, when firing, the actor consumes M_j tokens from its input buffer.

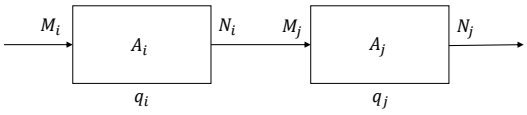

Fig. 2. Two actors in a dataflow model of computation

There exist dataflow models of different varieties [11]. In the synchronous dataflow (SDF) [12], the number of produced and consumed tokens are specified for each actor in advance. To avoid buffer overflow, for each pair of actors (A_i, A_j), the following balance equation must be satisfied:

$$q_i N_i = q_j M_j \tag{1}$$

where q_i, q_j are the firing rates of actors A_i and A_j respectively, N_i is the number of tokens actor A_i produces when firing, M_j is the firing rule for actor A_j.

In the *dynamic dataflow (DDF)*, the firing rules can specify different number of tokens for each firing, whereas in SDF the required number of input tokens is fixed. DDF is a fully dynamic model, i.e. all actor firing are determined at runtime. Structured dataflow enhances DDF with the structured programming concepts such as loops and conditions. The LabVIEW programming environment is one of the commercially available examples of structured dataflow implementations [11].

In [13], the authors propose a Parametrized Synchronous Dataflow (PSDF) model. PSDF tackles the modeling data-dependent dynamic DSP systems, but in addition allows constructing efficient quasi-static schedules. The latter are created as a result of compile-time analysis, and restrict the number of runtime decisions.

In [14], *Homogeneous Parametrized Dataflow (HPDF)* graphs are introduced as a means of image processing applications modeling. HPDF model is based on the *Parametrized Synchronous Dataflow (PSDF)*, with the restriction on the number of tokens along dataflow graph edges. To model typical image processing applications, in HPDF the data production and consumption rate is the same along dataflow graph edges. In [14], the vision algorithms of gesture recognition and face detection were modeled using HPDF, and in [15], the HPDF model was used for mapping a gesture recognition algorithms onto an FPGA board.

In [16], data flows in vision systems are proposed to be formalized with the means of meta-modeling, a method that allows to describe a system with the appropriate well-defined (machine-readable) language.

C. Communicating Sequential Processes

Communicating Sequential Processes (CSP) [17] is a process algebra formalism, used for modeling concurrency. A process behavior in CSP is defined as a trace of actions in which the process is engaged. The concurrent processes can communicate and share resources. The algebraic nature of CSP allows creating new processes from the existing ones.

In [18], the authors adapted the stream algebra formalism, used within the database community for modeling relational

streams, in order to tackle the problem of image stream processing. The proposed algebra includes stream operators for image processing and flow control, and is defined in terms of Communication Sequential Processes (CSP) and Stream Processing.

D. Supervisory control of discrete-event systems

To control a complex system, one needs to specify the operation sequence leading to the desired system's outcome. An *operation* in this context is a specification of an action or task executed by the system. This specification can be made on various levels of details. For example, in an automation system, operations can model a complete assembly of a product or sending a signal to an actuator [19]. In the Supervisory Control Theory [20], the controlled systems is modeled in a form of finite automata. A finite automaton consists of a set of states and a set of events leading to transition from one system state to another. To control a system modeled as a finite automaton, a discrete event supervisor is implemented. The latter, which plays a role of a closed-loop controller, responds to a sequence of asynchronous events and produces control output aimed at establishing the desired event sequence.

Based on the theoretical foundation of the Supervisory Control Theory and the formalism of Extended Finite Automata (EFA) [21], in [22], a graphical language dubbed SOP (Sequence of Operations) was proposed. Using SOP, the developer created individual self-contained operations, with all the product- and process-related logic expressed in the operations' pre- and post-conditions. The underlying EFA-based description was preserved, allowing for using the created models for formal verification and synthesis. As presented in Figure 3, an operation O_k can be formally described as an automaton with three states: initial state O_k^{init}, execution state O_k^{exec}, and finishing state O_k^{stop}. O_k begins its execution when the starting condition $C_k^{\uparrow}$ is satisfied and starting event $O_k^{\uparrow}$ occurs. The same logic holds for finishing condition $C_k^{\downarrow}$ and finishing event $O_k^{\downarrow}$.

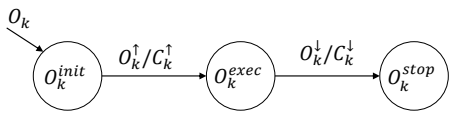

Fig. 3. Operation automaton

E. Limitations of the existing approaches

Dataflow models, especially the constrained SDF, PSDF and HPDF, are particularly suitable for describing hardware, such as digital signal processors and field-programmable gate arrays (FPGAs). This is the reason why the dataflow research is largely done within the embedded systems community. Though [14] and [15] apply dataflow modeling techniques to computer vision applications, the target use-case remains the design of image processing hardware, and therefore one requires strict formal properties, such as bounded memory requirements and efficient synthesis solutions [15], which restrict the expressiveness of the dataflow models. Dynamic dataflow models therefore seem more suitable for describing more complex data-dependent computer vision applications.

A limitation of dataflow in general is its reliance solely on data tokens as the only actor triggering mechanism. This may be sufficient for many vision applications and necessary for hardware specification. However, when industrial applications of vision are considered, one would benefit from the ability to apply event-based semantics in system design.

The formalisms and techniques of the Supervisory Control Theory allow to model and control system dynamics in terms of instantaneous discrete events. The EFA/SOP approach allows visualizing operation sequences in visually appealing way, somewhat similar to the actor diagrams. However, the signal paths and signals themselves are not represented in the discrete event systems frameworks.

Communicating Sequential Processes has proven to be a powerful formalism for describing concurrent systems, but they lack an expressive graphical representation, which may be poorly perceived by people outside the area of computer science, e.g. manufacturing engineers.

III. Formal specification of vision systems

A vision system is a system that acquires visual data and processes it to obtain a meaningful information or a new visual representation. A mono vision system uses one camera. A stereo vision system applies two or more cameras sharing the same view, which allows to reconstruct 3D information from the perceived scene. In the industrial context, vision systems are often used to control robots and various transportation systems. Therefore, it is required to be able to both describe the dataflow inside the vision system and consider the vision task in the context of a larger cyber-physical system.

The proposed approach, dubbed Discrete Event Dataflow (DEDF) extends the dynamic dataflow model with event-based semantics. It allows mapping between dataflow and discrete event models and accounting for typical signals conveyed in vision systems.

Before presenting the DEDF concept, the common vision data types are formally presented in terms of sets and signals.

A. Common signals and data types

Depending on the application, vision systems can process either individual images or video streams, where the latter constitute streams of distinct image frames. An image is often modeled as a signal in which an intensity is distributed over rectangular space. In this paper, a functional notation for signals is used. As described in [23], [24], any signal can be represented as a function f from domain D to co-domain C:

$$f : D \to C \tag{2}$$

An image acquired by the digital camera constitutes a two-dimensional array of pixels, each having (in case of a grayscale image) an intensity value in the range from 0 to 255. Thus, the range of intensity values can be defined as a

set $Intensity = \{0,1,...255\}$. A digital image with h rows and w columns can be formally defined as a signal from discrete image space $DISpace = \{0,1...h-1\} \times \{0,1,...w-1\} \subset \mathbb{Z} \times \mathbb{Z}$ to a set of intensity values [23], [25]:

$$Image : DISpace \rightarrow Intensity \tag{3}$$

In certain cases, continuous image space $CISpace = \{0,h-1\} \times \{0,w-1\} \subset \mathbb{R} \times \mathbb{R}$ needs to be considered, for example, for subpixel-precise measurements.

Function *Image* in (3) maps each tuple in the image space to the corresponding level of intensity. For the color images, the analogous function maps to the triple of intensities of red, green, and blue spectrum [23], [25]:

$$ColorImage : ImageSpace \rightarrow Intensity^3 \tag{4}$$

A subset of image space that is obtained as a result of image segmentation, is referred to as space relation. Depending on the application, one can consider discrete ($DSpaceRel \subset DISpace$) or continuous ($CSpaceRel \subset DISpace$) space relation. The graphical representation of image spaces and the correspoding space relations are provided in Figure 4.

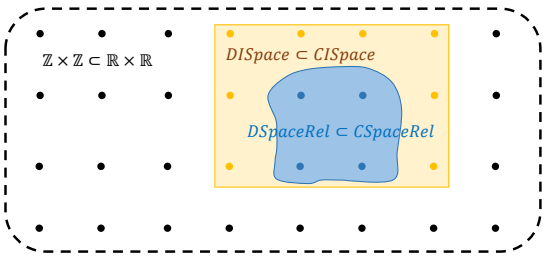

Fig. 4. Continuous and discrete space sets

A space relation does not convey any image-related information, and should therefore be distinguished from a region, that represents a part of the image belonging to the particular space relation:

$$Region : SpaceRelation \rightarrow Intensity \tag{5}$$

Geometric primitives can easily be formalized having the definition of continuous image space. For instance, a point $p \in CISpace$; a circle c, parametrized by its center and radius: $c \in CISpace \times \mathbb{R}$; an ellipse e, parametrized by the center and the major and minor radii: $e \in CISpace \times \mathbb{R}^2$.

B. Discrete Event Dataflow

A generic dataflow graph, as described in [26], can be formally represented as a triple G:

$$G = (A, L, P) \tag{6}$$

where $A = \{a_1, a_2, ...a_m\}$ is a set of actors, $L = \{l_1, l_2, ...l_n\}$ is a set of links, and $P \subseteq (A \times L) \cup (L \times A)$ is a set of ports.

The above representation is very similar to a Petri net: the links are treated as special-case graph vertices (analogous to actors, but responsible for data transfer), whereas ports connect actors to links (output ports, set $A \times L$) and links to actors (input ports, set $L \times A$).

The definition (6) is very basic and accounts only for the structure of the dataflow graph. The Discrete Event Dataflow (DEDF) is introduced in order to achieve a more expressive semantic.

DEDF can semantically be regarded as an extension of dynamic dataflow. It is aimed to account for the discrete event behavior, the tokens number specification, and actor-based event generation.

Discrete event behavior is important to consider in the context of industrial vision systems, since the controlled process is often modeled as a discrete event system. The concept of an actor can easily be mapped to the concept of operation, and it is therefore possible to associate each actor with an automaton model having three states and two events, see Figure 3.

The meaning of the number of tokens per dataflow graph port is analogous to the one in other models. In DEDF, however, the graph ports are divided into two sets: those with a fixed number of tokens, P_{const}, and those with variable number of tokens, P_{var}. For each port in P_{const}, the respective number of tokens should be specified. To assure predictable actors behavior, it is clear that the actors' input ports should be specified with only fixed number of tokens, i.e. $P_{const} \subseteq L \times A$.

Since DEDF is a dynamic and event-based model, the actor has the ability to initiate events. To express this ability, an event generator is introduced as a pair (a_i, e_j) meaning that actor a_i can initiate event e_j.

The Discrete Event Dataflow model is a 7-tuple:

$$G = (A, L, P_{const}, P_{var}, C, \Sigma, S) \tag{7}$$

where

$A = \{a_1, a_2, ...a_m\}$ is a set of actors

$L = \{l_1, l_2, ...l_n\}$ is a set of links

P_{const} is a set of ports with constant number of tokens, P_{var} is a set of ports where the number of tokens varies during runtime. $P = P_{const} \cup P_{var}$, with $P \subseteq (A \times L) \cup (L \times A)$ as the set of all ports

$C : P_{const} \rightarrow \mathbb{Z}$ is the vector/function specifying the number of tokens in each element of P_{const}

Σ is a set of events

S is a set of event generators, $S \subseteq A \times \Sigma$

An actor a_k in the dataflow model can be described as an operation O_k. The execution of O_k (state O_k^{exec}) is triggered by event $O_k^{\uparrow}$ when condition $C_k^{\uparrow}$ is satisfied. In the case of dataflow actor, this condition can be formulated as follows:

$$C_k^{\uparrow} = Fire_k \wedge Spec_k \tag{8}$$

where $Spec_k$ is a general application-specific condition for an operation, and $Fire_k$ is a condition specific to the actor/token semantics. Thus, if for actor a_k there exist m input ports $\{p_1, p_2, ...p_m\} \subset L \times A$, the firing condition is specified as follows:

$$Fire_k = (B_{p_1} = C_{p_1}) \wedge (B_{p_2} = C_{p_2}) \wedge \ldots \wedge (B_{p_m} = C_{p_m}) \quad (9)$$

where B is a vector/function specifying the current (at runtime) number of tokens per each port, i.e. $B : P \to \mathbb{Z}$, and C_{p_k} is number of tokens requirement per port k.

Definition (7) is generic in the sense that it does not specify the typing of the links and other characteristics. This means that DEDF can be applied in many different contexts by simply extending the model with appropriate vectors. For example, if typing information is important for the analysis, one can extend the DEDF graph G with a set of types T and vector/function specifying types for each link, i.e. $LinkTypes : L \to T$. In the same manner, it is possible to introduce some performance characteristics metrics for either actors or links.

IV. Case studies

A. Camera calibration

In most of the industrial cases, it is crucial to obtain measurements from a vision system that are expressed in real-world coordinates. This requires transforming pixel measures into metric values such as millimeters. To perform such transformation, the knowledge of the appropriate rigid transformations and intrinsic parameters of the cameras need to be obtained.

In this paper, the calibration method described in [27] and [28] is considered. The details of computing camera intrinsic parameters are omitted, and only the high-level view on the algorithm is presented. This calibration method, with certain modifications, is implemented in the OpenCV library, and can be described as follows:

1) The camera acquires an image of a rectangular planar calibration object with easily identifiable features, such as a chessboard.
2) All the corners of the chessboard are identified.
3) The corners identification results are filtered. If identification of the corners is unsuccessful, the results, along with the respective image, are discarded.
4) The previous steps are repeated until c_1 "good" chessboard corners results are obtained.
5) c_1 batches of chessboard corners results and the respective c_1 images are used for camera calibration.

The sequence above can be formally modeled as a DEDF depicted in Figure 5.

The *Operator* actor represents a human, supervising the calibration process, or some automated system. The vision system in this case is represented as a *Calibrator* actor, composed of a set of lower-level actors with self-explainable names: $A = \{AcquireIm, a_2, FindCorners, FilterCorners, Calibrate\}$. Actor a_2 is a splitter that sends the same signal to two different links. Each link possesses a type specified on each respective arc on the figure.

The types of tokens passed through the data flow include images (I), chessboard corners sets (*Corners*), and calibration results (*CalibRes*).

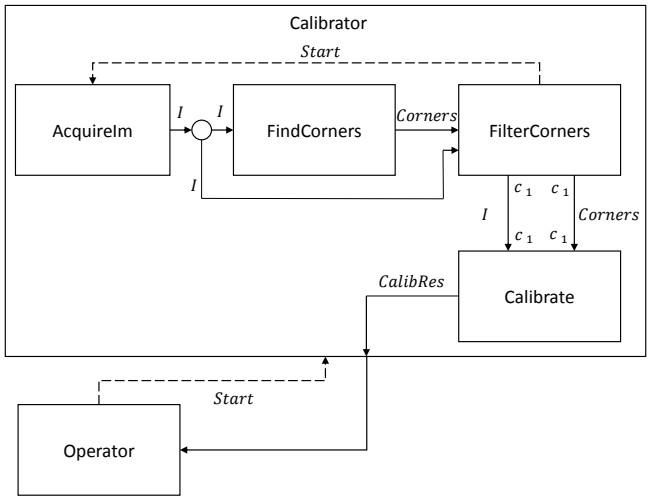

Fig. 5. DEDF model of camera calibration process

For each port $p \in P_{const}$, if $C_p \neq 1$, its number is specified on the diagram; if $C_p = 1$, the number 1 is omitted. In Figure 5, c_1 defines some constant specification. If a port $p \in P_{var}$, it is specified by a symbol v. In Figure 5, there is no such port that belongs to P_{var}.

Dashed lines specify event generators. In Figure 5, actor *FilterCorners* is able to generate event $AcquireIm^{\uparrow}$ if there has not been c_1 "good" images with the identified corners accumulated, and actor *Operator* invokes the start of the calibration process by generating $Calibrator^{\uparrow}$. In both cases, the events belong to the respective actors.

B. Passive stereo vision

The passive stereo vision process uses two images of the same scene acquired at the same moment by two or more cameras to reconstruct the 3D coordinates of points of interest. The reconstruction is done by triangulation of the matching pairs of the identified features, having the rigid transformation between two cameras and the cameras' intrinsic parameters [29].

Figure 6 shows a high level DEDF model of a passive stereo vision system. A composite actor *StereoMatcher* is started by an external system. Within *StereoMatcher* there are two source actors - $AcquireIm_1$ and $AcquireIm_2$- those having no input ports. This implies that when event $StereoMatcher^{\uparrow}$ occurs, it leads to immediate occurrence of both ${AcquireIm_1}^{\uparrow}$ and ${AcquireIm_2}^{\uparrow}$.

When the images from each cameras have been acquired, they are processed by an algorithm in $FindFeatures_1$ and $FindFeatures_2$ that extracts the features of interest. The triangulation is performed by actor *Compute3D* that is parametrized by stereo vision systems parameters from the *ExternalSystem*. A variable number of 3D points is then fed back to the *ExternalSystem*.

V. Discussion and conclusion

Motivated by the increasing complexity of the contemporary industrial systems and the challenges in implementing

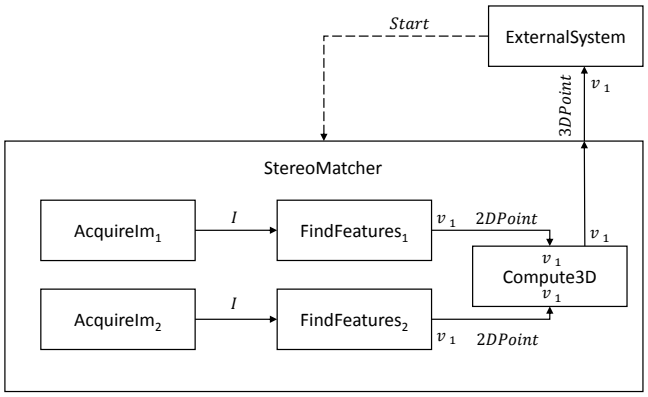

Fig. 6. DEDF model of a passive stereo system

machine vision applications, a formal approach, Discrete Event Dataflow (DEDF), was presented in this paper. DEDF aims at capturing both the nature of streaming applications and the discrete event behavior of larger industrial systems. The DEDF formalism is generic and can be used for modeling applications other than vision, such as digital signal processing, sensor fusion, and general data flow.

A graphical representation of DEDF was presented with two examples, camera calibration and passive stereo vision system. It is important to note, however, that a complete specification of DEDF diagrams is yet to be developed, which includes the graphical representation of actor-independent events.

More thorough formal analyses of DEDF ought to be performed in the future. Because dataflow graphs are in many respects analogous to Petri nets, it is of interest to conduct a comparison study between these two formalisms, with the addition of events semantics. In addition, the analysis algorithms for DEDF are yet to be developed. Another aspect that has to be clarified in the future is the semantics of feedback data flow.

The practical side of DEDF in general and implementation of vision systems in particular was not considered in this paper. An appropriate software framework, together with the additional synthesis methods are required.

The idea of variable number of tokens per link was presented. However, in reality it is useful to be able to constrain this number with a specified limit. Therefore, a constraint extension of DEDF shall be formally defined in the future.

REFERENCES

[1] C. Steger, M. Ulrich, and C. Wiedemann, *Machine Vision Algorithms and Applications*, ser. Wiley-VCH Textbook. Wiley-VCH, 2007.

[2] E. N. Malamas, E. G. M. Petrakis, M. Zervakis, L. Petit, and J.-D. Legat, "A survey on industrial vision systems, applications and tools," *Image and Vision Computing*, vol. 21, no. 2, pp. 171–188, 2003.

[3] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision*. Thomson-Engineering, 2007.

[4] H. Golnabi and A. Asadpour, "Design and application of industrial machine vision systems," *Robotics and Computer-Integrated Manufacturing*, vol. 23, no. 6, pp. 630–637, 2007.

[5] J. Campos, C. Seatzu, and X. Xie, *Formal Methods in Manufacturing*, ser. Industrial Information Technology. CRC Press, 2014.

[6] S. Kalpakjian and S. Schmid, *Manufacturing Engineering & Technology*, 7th ed. Pearson Prentice Hall, 2013.

[7] M. Santochi and G. Dini, "Sensor technology in assembly systems," *CIRP Annals - Manufacturing Technology*, vol. 47, no. 2, pp. 503–524, 1998.

[8] E. A. Lee, S. Neuendorffer, and M. J. Wirthlin, "Actor-oriented design of embedded hardware and software systems," *Journal of Circuits, Systems and Computers*, vol. 12, no. 3, pp. 231–260, 2003.

[9] P. Derler, E. A. Lee, and A. Sangiovanni Vincentelli, "Modeling cyber-physical systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, 2012.

[10] R. Stephens, "A survey of stream processing," *Acta Informatica*, vol. 34, pp. 491–541, 1997.

[11] E. A. Lee, S. Neuendorffer, and G. Zhou, "Dataflow," in *System Design, Modeling, and Simulation using Ptolemy II*, C. Ptolemaeus, Ed. Ptolemy.org, 2014. [Online]. Available: http://ptolemy.eecs.berkeley.edu/books/Systems/chapters/Dataflow.pdf

[12] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[13] B. Bhattacharya and S. S. Bhattacharyya, "Parameterized dataflow modeling for dsp systems," *IEEE Transactions on Signal Processing*, vol. 49, no. 10, pp. 2408–2421, 2001.

[14] M. Sen, S. S. Bhattacharyya, T. Lv, and W. Wolf, "Modeling image processing systems with homogeneous parameterized dataflow graphs," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. V, 2005, pp. 133–136.

[15] M. Sen, I. Corretjer, F. Haim, S. Saha, J. Schlessman, T. Lv, S. S. Bhattacharyya, and W. Wolf, "Dataflow-based mapping of computer vision algorithms onto fpgas," *EURASIP Journal on Embedded Systems*, vol. 2007, pp. 1–12, 2007.

[16] P. Trojanek, M. Stefańczyk, and T. Kornuta, "Modelling of data flow in component-based robot perception systems," *Pomiary Automatyka Robotyka*, no. 2/2013, pp. 260–265, 2013.

[17] C. A. R. Hoare, *Communicating sequential processes*. Prentice Hall International, Incorporated, 1985.

[18] M. A. Helala, K. Q. Pu, and F. Z. Qureshi, "A stream algebra for computer vision pipelines," pp. 800–807, 2014.

[19] K. Bengtsson, P. Bergagard, C. Thorstensson, B. Lennartson, K. Åkesson, Y. Chengyin, S. Miremadi, and P. Falkman, "Sequence planning using multiple and coordinated sequences of operations," *Automation Science and Engineering, IEEE Transactions on*, vol. 9, no. 2, pp. 308–319, 2012.

[20] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Springer, 2008.

[21] M. Sköldstam, K. Åkesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," in *Decision and Control, 2007 46th IEEE Conference on*, 2007, Conference Proceedings, pp. 3387–3392.

[22] B. Lennartson, K. Bengtsson, Y. Chengyin, K. Andersson, M. Fabian, P. Falkman, and K. Åkesson, "Sequence planning for integrated product, process and automation design," *Automation Science and Engineering, IEEE Transactions on*, vol. 7, no. 4, pp. 791–802, 2010.

[23] E. A. Lee and P. Varaiya, *Structure and Interpretation of Signals and Systems*, 2nd ed. LeeVaraiya.org, 2011.

[24] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-physical Systems Approach*. Lulu.com, 2011.

[25] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. John Wiley & Sons New York, 2006.

[26] K. Kavi, B. Buckles, and U. Bhat, "A formal definition of data flow graph models," *Computers, IEEE Transactions on*, vol. C, no. 11, pp. 940–948, 1986.

[27] Z. Zhang, "A flexible new technique for camera calibration," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 11, pp. 1330–1334, 2000.

[28] P. F. Sturm and S. J. Maybank, "On plane-based camera calibration: A general algorithm, singularities, applications," in *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, vol. 1, 1999, Conference Proceedings, p. 437 Vol. 1.

[29] Z. M. Bi and L. Wang, "Advances in 3d data acquisition and processing for industrial applications," *Robotics and Computer-Integrated Manufacturing*, vol. 26, no. 5, pp. 403–413, 2010.