Procedia CIRP 97 (2020) 453–458

Contents lists available at ScienceDirect

Procedia CIRP

journal homepage: www.elsevier.com/locate/procir

Development of application programming interface prototype for injection molding machines

Olga Ogorodnyk[a,*], Mats Larsen[b], Kristian Martinsen[a], Ole Vidar Lyngstad[b]

[a] Department of Manufacturing and Civil Engineering, Norwegian University of Science and Technology, Teknologivegen 22, 2015 Gjøvik, Norway
[b] SINTEF Manufacturing AS, P.O. Box 163, 2831 Raufoss, Norway

ARTICLE INFO

Keywords:
Injection moulding
Application programming interface
Cyber-Physical Systems
Industry 4.0

ABSTRACT

This paper describes architecture and development of an open application programming interface (API) prototype for injection molding machines (IMMs), useable for sensor and machine/process data logging and setting necessary process parameter values. The API is based on PCMEF (presentation, control, domain and foundation) architectural framework and OSI 7 layer communication model. The interface allows to retrieve values of up to 97 machine and process parameters. It also includes a module for acquisition of data from additional sensors such as pressure and temperature sensors installed in the mold. Industrial Raspberry Pi (RevPi) is used to perform analog-to-digital signal conversion and makes sensors data accessible via the API. Logging of different parameters from the machine and from sensors is synchronized and sampling frequency can be adjusted if necessary. Depending on chosen frequency, the system can provide real-time or soft real-time communication. The interface allows to build a distributed computer-based system, which gives benefit over the use of a PLC system with respect to Industry 4.0 standards.

1. Introduction

Several studies highlight the importance of collection and analysis of process data from manufacturing processes (Vrabič et al., 2017). The increased attention to the concepts of Industry 4.0, cyber-physical systems (CPS), self-optimizing systems, as well as so-called data-driven methods for creation of models for prediction and classification (Yin et al., 2014) underline this. Although CPS is still mainly a concept (Kull, 2015, Saldivar et al., 2016) and is far from industrial implementation (Lee et al., 2014), we see a trend of machine tool suppliers offering more sensors, data collection and simulation software. Within the CPS concept lies the potential of optimization of production system behavior in real time (Negri et al., 2017). To be able to achieve this the systems need to provide robust and synchronized data acquisition, storage and processing, external communication and intelligent process control (Vrabič et al., 2017, Lee et al., 2014, Tellaeche and Arana, 2013). Models created by data-driven methods, such as machine learning (ML) can provide necessary flexibility and robust adaptation in the changing production environment (Yin et al., 2014).

To reach these goals there is a need to develop extended and commonly accessible application programming interfaces (APIs) between machine tools and external systems. Such APIs must enable real-time process data collection and processing. In the case of injection molding this would include logging of data from sensors installed in the injection molding machines (IMMs) and molds. The obtained data can be further used for optimizing process parameters, process monitoring and control.

1.1. Injection molding challenges towards industry 4.0

Quality of injection molded parts are dependent on many parameters: product design, material properties, the quality of the mold, machine tool parameters such as holding pressure, backpressure, cooling time, injection speed (Ogorodnyk and Martinsen, 2018, Zhao et al., 2014). Due to this, selection of the parameter settings that enable production of high quality parts has been an important research area over the years (Huang and Tai, 2001, Kashyap and Datta, 2015). Similar to other industrial machines, modern IMMs can be equipped with manufacturing execution systems (MES) which include a data logging function, usually providing monitoring of machine status, remote access to machine setup, data logging of machine and process parameters and display-

* Corresponding author.
E-mail address: olga.ogorodnyk@ntnu.no (O. Ogorodnyk).

https://doi.org/10.1016/j.procir.2020.07.005

ing the data in a feasible way. Examples are ARBURG's "Host computer system (ALS)" (ARBURG. Host computer system (ALS) 2020) and MES HYDRA (MDPV 2020). These are definitely a step towards Industry 4.0, but there are still challenges such as real-time data collection, as many of the MES log parameter data only once per production cycle, as well as the absence of possibility to synchronize data acquisition from machine built-in and additional sensors installed, for example, in a mold. One of the main reasons for this is that the available systems have a closed architecture and predefined functionality without an open API.

In addition, selection of a proper communication protocol to establish communication between an IMM and a PC or another data storage and processing unit is important. EUROMAP 63 protocol was developed in 2000 and uses file based data transfer (EUROMAP 2020). EUROMAP77 was released in 2018 and is based on OPC/UA but is not directly connected to IMM's control loop and might not be able to guarantee real-time or soft real-time communication. In addition, some IMM suppliers such as ENGEL provide their own data exchange interfaces (ENGEL machine interface (EMI)). All of these protocols need to be considered if the API or an MES is to be used with various types of IMMs.

Research literature, at the same time, offers examples of data acquisition during the injection molding process, such as in-mold force (Tellaeche and Arana, 2013), mold temperature, screw displacement and velocity data (Zhao et al., 2014, Zhou et al., 2017). Zhang Y, Mao T (Zhang et al., 2016) are using a hydraulic pressure sensor to estimate nozzle pressure values and increase controllability of the process. In (Charest et al., 2018) pressure transducers, thermocouples, velocity, position and flow sensors are installed to acquire IMM data. However, in these examples the data is acquired from the sensors additionally installed in the IMM. Ironically, a lot of this data could be acquired from sensors already installed in the machine by its manufacturer, but due to restriction of commercial software and hardware, new sensors needed to be installed to get easy access to data and analyze it.

In this paper, on the other hand, development and application of an open API prototype applied to an IMM and in-mold sensors is presented. Such API can be used for industrial and research application in order to address the following tasks:

- Ability to use the API with various types of IMMs of different manufacturers and communication protocols;
- Acquisition of IMM machine and process parameters data from built-in sensors;
- Acquisition of IMM machine and process parameters data from additional sensors;
- Synchronized acquisition of data from built-in and additional sensors;
- Setting of machine and process parameters on the IMM without hindering the IMM operational security;
- Possibility to add custom modules on top of the initially available functionality. Such modules might include implementation of various algorithms for data analysis, parameters optimization and intelligent process control.

2. Development of a generic API

Application programming interface is a software product that includes number of clearly defined methods for communication between different components, in our case, between an IMM, a PC and a data acquisition system connected to mold sensors. As an extension to the tasks presented in previous section, the API needs to comply with the following requirements:

- Compatible with personal computers.
- Open with possibility to build on existing functionality.
- Soft real-time logging sampling rate faster than 2 Hz.

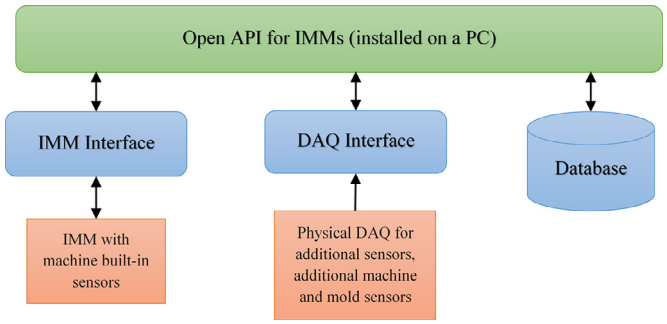

Fig. 1. API modules.

- Real-time logging for sampling rate slower and equal to 2 Hz.
- Logging of up to 97 machine and process IMM parameters.
- Logging of data from additionally installed sensors.
- Ensured synchronized acquisition of data from machine built-in sensors and additionally installed sensors;
- Option of setting values of machine and process parameters, where allowed, without overriding the current value if the entered value is not acceptable due to being too high or too low for secure operation of the IMM in use.

Python3 programming language was chosen to create a prototype of the proposed application programming interface. The API's architecture is designed based on PCMEF (Maciaszek and Liong, 2004, Kurose and Ross, 2010) *layered architectural framework* providing easy interaction with the IMM. Benefit of this framework is its stability and modularity, which is possible since only downward dependencies between the layers are allowed. As a result, *"changes in higher layers don't create a cascade of modifications in lower layers"* (Madeyski and Sochmialek, 2005). The framework consists of four main layers: *presentation, control, domain* and *foundation* (Maciaszek and Liong, 2004).

Communication design of the proposed application programming interface was created following OSI 7 layer model. It is Open Systems Interconnection model that generally includes the following levels: *application, presentation, session, transport, network, link* and *physical* (Kurose and Ross, 2010). The general communication design of ENGEL EMI data exchange protocol used in the API's server-side is also structured after OSI 7 layer model. The server-side is implemented by ENGEL and includes all 7 layers of the model. The authors of the paper concentrated on the client-side implementation which includes layers number 5, 6 and 7 of the OSI model. Our *Application layer* is with "get and set methods" for machine parameters. Our *Presentation layer* includes Python's XML-byte stream implementation for data interpretation from the IMM to construct an XML-byte stream. This layer must guarantee readability of server on the IMM, meaning that server has to react on request by a method defined in the XML-byte stream. The *Session layer* is based on socket Python library and is multi-client compatible. The developed API includes three main modules: *IMM interface, Data acquisition system for additional sensors (DAQ Interface)* and *database*, as shown on Fig. 1. Here the general API is depicted as a green block, while three software modules it includes are in blue blocks, while the orange blocks include hardware that the API and its modules are connected to. *IMM interface* is part of the API that is used for establishment of connection with injection molding machine in use and requesting and setting corresponding parameter values. *DAQ interface* is used for acquisition of data from any additionally installed machine and mold sensors. *Database* is needed for organized storage and easy access to the sampled data. If necessary, additional modules for testing data processing algorithms and IMM control routines can be added.

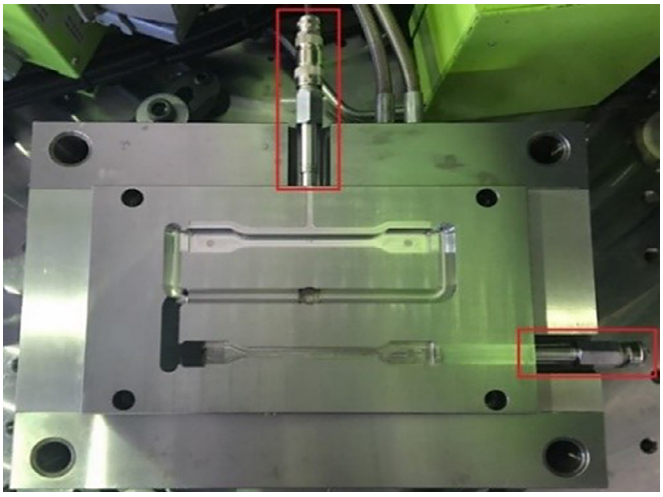

Fig. 2. Mold for production of dog bone specimens.

3. API application on ENGEL IMM

The developed API prototype was tested with an ENGEL insert 130 vertical IMM with CC300 control unit. Focus parts are dog bone specimens for tensile stress testing with 170 mm length, 20 mm width and 4 and 15 mm thickness, according to ISO 527-2 (ISO 2012). The mold for production of the 15 mm thick parts is shown on Fig. 2. It has two Kistler 4021B10H1P1 multi-sensors installed for monitoring mold temperature and pressure.

We have selected Raspberry Pi RevPi Core3 and RevPi Analog Input Output (AIO) modules for the digital sampling of the signals. The modules are connected through PiBridge. RevPi allows performing Delta Sigma Conversion using analog-to-digital converter model ADS1248 with 24-bit resolution. Unlike commercial DAQs from manufacturers like NI and HBM, RevPi is an open, modular industrial PC which provides flexibility of software alternatives (RevPi - Industrial Raspberry Pi 2020). The sampling rate is unfortunately low, in practice about 125 Hz due to load on the PiBridge. This will lead to about 5 ms update time on the PiBridge for each AIO module connected, which is, however, sufficient for our case. The computer networking device that is used to connect all the devices together is a 1000 Mbps switch.

3.1. Data acquisition from machine built-in sensors

The IMM interface is the API module responsible for getting and setting parameter values from machine built-in sensors. The main tasks of this module are establishment of connection with the IMM, periodical checks that the connection is up and running, reading of requested machine and process parameters data and its storage in a first-in-first-out (FIFO) queue.

The ENGEL Insert 130 IMM used in this study supports EUROMAP63 and ENGEL Machine Interface (EMI) data exchange protocols. To satisfy the need for soft real-time and real-time data exchange, EMI data exchange protocol is used in the API, as due to the authors' experience it is much faster than the EUROMAP63. This module includes an interface class specifically designed for data exchange with the help of EMI data exchange protocol and cannot be used as a general interface class for all types of data exchange protocols. If, for example, EUROMAP77 needs to be implemented, a new interface class has to be added to follow corresponding protocol specifications. In private communication with ENGEL the Uniform Resource Identifiers (URIs) necessary for accessing different values on the IMM were obtained.

The module uses Pyro4 (Python Remote Objects), "xml.etree.ElementTree", socket, thread, queue and datetime Python libraries. Pyro4 library is used to enable distributed computing between network-based objects, as Pyro4 is able to handle automatic locating of remote calls based on the URIs. The "xml.etree.ElementTree" library was used to construct XML byte-streams and for the conversion of byte-streams into strings. This is necessary since EMI is based on XML exchange between IMM and a PC. The socket library, in its turn, is used to establish a client-server connection between a PC and the IMM, as EMI uses TCP/IP protocol for communication. Thread library is necessary for the development of high-level threaded interfaces, while queue library *"implements multi-producer, multi-consumer queues"* (PythonSoftwareFoundation 2020) and is useful for threaded programming and safe data exchange between different threads. Datetime library, on the other hand, is used to retrieve machine date and time and ease operations with it.

The API prototype can log data in three different modes *"Idle"*, *"Fixed cycles"* and *"Flexible cycles"* depending on the application. In the *"Idle"* mode controller ensures that connection is up and that application is able to access the interface's methods for asynchronous interaction, while the *"Fixed cycles"* mode is a periodical sampling mechanism of parameters from the IMM that samples all of the parameters simultaneously. The sampling results are available in the queue for the application process. *"Flexible cycles"* mode takes care of the unbalanced update sampling frequencies to avoid oversampling. Machine parameters on the IMM can have different update frequencies depending on the sampling limitations of the data acquisition system on the IMM. These frequencies can also be manipulated by the user via the IMM's general user interface. The updating frequencies typically vary from 10 to 1000 ms. At the first run in the *"Flexible cycles"* mode, the interface acquires sampling frequency of each parameter from the IMM and groups parameters that have the same update rate. In this mode, interface checks the last datetime (using datetime Python library), when each parameter group was updated, and determines if it is time to acquire a new portion of data from the IMM.

Another function of the module is to load user-requested parameters provided in a .csv file. Data from the file is loaded into a "ParameterList" parameter and is then used to specify the mapping between Uniform Resource Identifiers of different parameters on the IMM and set, as well as actual parameter values. The description of "ParameterList" structure is provided in Table 1.

3.2. Data acquisition from additional sensors

The logging of data from both built-in and additional sensors is synchronized. To do this API uses one of the deployed methods called *event_sample()*. It needs to be triggered for both IMM and the data acquisition system in use (in our case RevPi). The data is logged only when the machine mold is closed in order to minimize the amount of memory used for data storage. In this case, the data acquisition of both IMM parameters and additional sensors parameters starts simultaneously when the IMM mold closes. To detect the mold closing, a *"mold force"* parameter is used. The software unit waits until the mold force parameter reaches a value of $\geq$ 300 kN and then starts data collection. It stops when the mold force parameter value becomes lower than the above-specified value. Each injection molding machine has a similar parameter to detect the mold closing, therefore, synchronization based on this method is robust enough to use it with IMMs of different types and manufacturers. The list of obtained parameter values is then provided in a .csv, .json or .pickle file format saved in a specified by a user folder.

This research included the use of a mold with two pressure and temperature multi-sensors. This is what the DAQ module of the

Table 1
"ParameterList" description.

Column name	Column name description	Example
name	Name of a specific parameter, name defined by user.	Plasticizing_time
path_act_value	Mapping to a predefined URI address on the IMM. This URI returns actual value of the specific parameter given by the column "name".	URI for Plasticizing time actual value
path_set_value	Mapping to a predefined URI address on the IMM. This URI is setting user-given value of the specific parameter given by the column "name".	URI for Temperature in heating cylinder set value
enable	Defines if the specific parameter given by the column "name" is enabled or disabled.	0 or 1
description	User's description of a parameter.	Plasticizing time of the molding process
unit	Predefined unit of a parameter.	s

Table 2
Statistics of overhead measurement for RevPi.

Sampling rate [Hz]	Mean [s]	Std. dev. [s]	Max. [s]	Min. [s]	Overhead over 0.025 s [%]
100	0.015363	0.020718	0.467	0	0.96
50	0.006215	0.024150	0.262	−0.01	1.29
33	0.000007	0.0241035	0.448	−0.2	1.14
2	0.000603	0.0007935	0.006	−0.004	0.00

proposed API is needed for. However, the API can also be utilized with other types of additional sensors.

As described previously, Industrial Raspberry Pi (RevPi) consisting of two modules, RevPi Core3 and RevPi AIO, was used as a physical data acquisition system for this purpose. A simple Python programming language script is used to convert sensor signals from analog to digital and synchronize logging with data acquisition from the machine built-in sensors to provide data integrity. This is significantly easier than using DAQs from National Instruments (NI) or HBM to acquire sensor data, as they require the use of commercial "black box" software that needs to be additionally purchased such as, for example, LabView or CatMan, while Python interpreters are usually open source and free of charge.

3.3. Database

Currently, data from each production cycle is written to a separate file of a user-specified type (.csv, .json, .pickle) and is stored in a folder chosen by a user on the PC connected to the IMM. A proper database management unit needs to be developed and added to the system in future versions, providing a better way of storing big amounts of process data. This unit will enable users to create, structure and update a database with necessary parameters, as well as to structure larger datasets for their further analysis.

4. Validation of the developed prototype

Before using the API for acquisition of production or experimental data, it is necessary to investigate whether the developed API prototype complies with the specified real-time and soft real-time requirements.

All modules of the API were installed on the RevPi, where three different processes needed to be handled: handling the IMM, handling Kistler sensors signals and synchronization of data acquisition from the injection molding machine and mold sensors. All tests were conducted at length of 4900 samples for sampling rates at 100, 50, 33 and 2 Hz. The experiments measure the latency/overhead compared to the desired sampling rate.

The results of the statistical summaries of the overhead distribution for RevPi are presented in Table 2 and summaries for IMM's DAQ in Table 3. In the table, the "Sampling rate" column contains the used sampling rate value, "Mean" is the average of the overhead in seconds, "Std. dev" is the standard deviation in seconds, "Max" is the highest overhead time in seconds, "Min" is the lowest overhead time and "Overhead over 0.025s" is the percentage of samples with overhead over 0.025 s. The used sampling rate is subtracted from the period between samples, *(sample_n + 1 - sample_n) - sampling_rate* in the results to calculate the overhead/latency.

The results shown in Table 2 and 3, show that real-time restrictions of the system apply. For the test with 100 Hz sampling rate, the mean values are around 0.0154 for sampling on both IMM and the RevPi. This means that the system cannot comply with the given sampling rate. 10 ms logging frequency and an average overhead of 0.015366 provide 39.4 Hz real sampling rate.

Examining results from sampling at 50 Hz, it is possible to see that the system performs better than with 100 Hz sampling rate. In this case, the overhead mean value is around 0.0062 s (for both IMM and the RevPi) which gives a system response at 38.1 Hz. Even better results can be seen at sampling rate at 33 Hz. Here, the average overhead tends to zero, while the number of overhead values over 0.025% is 1.14 and 1.73 for the RevPi and the IMM's DAQ respectively. This indicates that 33 Hz can be one of the appropriate sampling rates for the proposed API.

2 Hz sampling rate has the overhead mean value for both IMM and the RevPi at around 0.0006, this is significantly lower than with 100 and 50 Hz sampling rates, but slightly higher than with the 33 Hz sampling rate. Regarding real-time demands, results show that the API is not completely predictable with a maximum overhead of 0.14 s and a lowest overhead/latency at −0.139 s on the IMM when sampling with the 2 Hz rate. This happens because the system tries to compensate for the deviation. On the IMM there are 2.34% of samples that have the overhead value higher than 0.025 s, while on the RevPi there are none. The RevPi demonstrates that it is able to perform real-time sampling with 2 Hz or lower sampling rate and soft real-time on higher speeds.

The API's unpredictability comes, among other things, from Python's memory management mechanism. It uses reference counting collector and generational garbage collector, known as *"gc module"* for memory reclaim (PythonSoftwareFoundation. gc - Garbage Collector interface 2020). Unlike many other languages, it does not necessarily release the memory back to the operating system, but instead keeps some parts of already allocated memory for use in the future. At the same time, it is possible to see that the IMM process is more unpredictable than the RevPi process. This might be caused by necessity to establish the server/client connection with the IMM, while the RevPi is directly connected to the sensors it acquires the data from.

Table 3
Statistics of overhead measurement for IMM.

Sampl. rate [Hz]	Mean [s]	Std. dev. [s]	Max. [s]	Min. [s]	Overhead over 0.025 s [%]
100	0.01537	0.014904	0.341	0.007	1.67
50	0.006229	0.017871	0.217	−0.003	2.43
33	0.000007	0.020911	0.408	−0.012	1.73
2	0.000604	0.013165	0.140	−0.139	2.34

4.1. Case study

The described API prototype was tested during two experiments. The first experiment was injection molding of two dog bone specimens per cycle with 4 mm thickness. It was based on Design of Experiment (DOE) created using Latin Hypercube sampling method (LHS) (Seaholm et al., 1988) that consisted of 32 combinations of the following parameters: holding pressure, holding pressure time, backpressure, cooling time, injection speed, screw speed, barrel temperature and mold temperature. Each of these combinations was launched five times, resulting in 160 machine runs and corresponding production cycles. During this experiment, values of 41 machine and process parameters were logged. The chosen sampling rate was 2 Hz.

The second experiment was focusing on molding dog bone parts with 15 mm thickness. Only one of the sensors in the mold was used for logging the data, as part of the cavity was closed to produce only one specimen per production cycle. Here DOE was created using the same LHS method and included 24 combinations of backpressure, cooling time, holding pressure, holding pressure time, injection speed and screw speed parameters. Each combination was launched three times, so the experiment consisted of 72 production cycles. This time 65 machine and process parameters were logged, as well as temperature and pressure from one of the sensors installed in the mold. The data from the first experiment has then been used in order to test application of ML methods to create prediction models for the quality of dog bone parts, more information on how this was done is provided in (Ogorodnyk et al., 2018).

5. Limitations and future work

As it is shown in the previous section the prototype API is able to comply with the defined requirements. Due to its openness it allows the user to access the necessary process parameters and log them with a chosen sampling rate. In addition, it provides possibility to add the necessary modules and classes for use of other communication protocols or rapid prototyping of the data analysis algorithms of interest. Anyone familiar with the Python3 programming language can deploy additional modules to the API prototype using the package provided in the following repository: https://github.com/SintefManufacturing/IMM_API.

The current API prototype version includes the following limitations:

- Only class for the EMI data exchange protocol is implemented;
- Real-time logging is provided on the speed of 2 Hz or lower, while the faster sampling speeds lead to the soft real-time data acquisition;
- The setting function of the API implementation is limited by the available parameter URIs and some of the parameter values can not be set through the API due to this;
- The API prototype doesn't include any graphical user interface;
- The API does not consider the information security aspects.

As a result, future work should include development of classes for other data exchange protocols, such as EUROMAP77 to facilitate use of the interface prototype with IMMs of other manufacturers. It is of interest to test the API prototype with other types of IMMs, data acquisition systems and additional sensors. At the same time, the database module needs to be developed further to allow storage of all the necessary process data in a corresponding database. Development of a graphical user interface (GUI) for the API is also of interest to make it more user friendly. Moreover, in the following versions the system's security needs to be considered and extensively worked on. In order to improve functionality and performance of the open API for IMMs and create a finished product rather than a prototype, collaboration with IMM manufacturers, as well as OPC/UA working group would be highly beneficial.

6. Conclusion

This paper has provided requirements, description of the development and capabilities of the prototype of an open application programming interface for injection molding machines. The interface is open for external interaction with the machine controller, allowing logging and setting values of process parameters. The openness of the prototype API also provides possibilities for rapid algorithm prototyping and testing, when developing control strategies in the laboratory or at a production site. The API's capability of complying with soft real-time and hard real-time data acquisition was tested with different sampling rates, and later the injection molding process data was logged during two different experiments. The data from the first experiment has later been used for application of ML methods for prediction of the produced parts quality.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research is funded by Norwegian Research Council as part of the "MegaMould" Project (project number: 256819) and through the SFI Manufacturing Project (project number: 237900).

References

ARBURG. Host computer system (ALS). 2020 [cited 2020 13.07.2020]; Available from: https://www.arburg.com/en/products-and-services/injection-moulding/production-management/host-computer-system-als/.

Charest, M., Finn, R., Dubay, R., 2018. Integration of artificial intelligence in an injection molding process for on-line process parameter adjustment. In: 2018 Annual IEEE International Systems Conference (SysCon),. IEEE, pp. 1–6.

EUROMAP. EUROMAP 77 – data exchange interface between injection moulding machines and MES. 2020 [cited 2020 25.06.2020]; Available from: https://opcfoundation.org/markets-collaboration/plastics-and-rubber-machinery/.

Huang, M.-C., Tai, C.-C., 2001. The effective factors in the warpage problem of an injection-molded part with a thin shell feature. J. Mater. Process. Technol. 110 (1), 1–9.

ISO. ISO 527-2:2017 Plastics – Determination of tensile properties – Part 2: test conditions for moulding and extrusion plastics. 2012 [cited 2020 13.07.2020]; Available from: https://www.iso.org/standard/56046.html.

Kashyap, S., Datta, D., 2015. Process parameter optimization of plastic injection molding: a review. Int. J. Plast. Technol. 19 (1), 1–18.

Kull, H., 2015. Mass Customization: Opportunities, Methods, and Challenges for Manufacturers. Apress.

Kurose, J.F., Ross, K.W., 2010. Computer Networking: a Top-Down Approach. Pearson Addison Wesley.
Lee, J., Kao, H.-.A., Yang, S., 2014. Service innovation and smart analytics for industry 4.0 and big data environment. Proc. CIRP 16, 3–8.
Maciaszek, L., Liong, B.L., 2004. Practical Software Engineering: An Interactive Case-Study Approach to Information Systems Development. Pearson Addison Wesley.
Madeyski, L., Sochmialek, M., 2005. Architectural design of modern web applications. Found. Comput. Decis. Sci. 30 (1), 49–60.
MDPV. Manufacturing Execution System HYDRA. 2020 [cited 2020 03.07.2020]; Available from: https://www.mpdv.com/en/products-solutions/mes-hydra/#c2077-1.
Negri, E., Fumagalli, L., Macchi, M., 2017. A review of the roles of digital twin in CPS-based production systems. Proc. Manuf. 11, 939–948.
Ogorodnyk, O., et al., 2018. Application of machine learning methods for prediction of parts quality in thermoplastics injection molding. In: International Workshop of Advanced Manufacturing and Automation. Springer, pp. 237–244.
Ogorodnyk, O., Martinsen, K., 2018. Monitoring and control for thermoplastics injection molding a review. Proc. CIRP 67, 380–385.
PythonSoftwareFoundation. queue — A synchronized queue class. 2020 [cited 2020 13.07.2020]; Available from: https://docs.python.org/3/library/queue.html#module-queue.
PythonSoftwareFoundation. gc - Garbage Collector interface. 2020 [cited 2020 13.07.2020]; Available from: https://docs.python.org/3/library/gc.html.
RevPi - Industrial Raspberry Pi. 2020 [cited 2020 25.06.2020]; Available from: https://revolution.kunbus.com/revolution-pi-series/.
Saldivar, A.A.F., et al., 2016. Attribute identification and predictive customisation using fuzzy clustering and genetic search for industry 4.0 environments. In: 2016 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA). IEEE, pp. 79–86.
Seaholm, S.K., Ackerman, E., Wu, S.-.C., 1988. Latin Hypercube Sampling and the sensitivity analysis of a Monte Carlo epidemic model. Int. J. Biomed. Comput. 23 (1–2), 97–112.
Tellaeche, A., Arana, R., 2013. Rapid data acquisition system for complex algorithm testing in plastic molding industry. Int. J. Mech., Aerosp., Ind., Mechatron. Manuf. Eng. 7 (7), 1391–1395.
Vrabič, R., Kozjek, D., Butala, P., 2017. Knowledge elicitation for fault diagnostics in plastic injection moulding: a case for machine-to-machine communication. CIRP Ann. 66 (1), 433–436.
Yin, S., et al., 2014. A review on basic data-driven approaches for industrial process monitoring. IEEE Trans. Ind. Electron. 61 (11), 6418–6428.
Zhang, Y., et al., 2016. A statistical quality monitoring method for plastic injection molding using machine built-in sensors. Int. J. Adv. Manuf. Technol. 85 (9–12), 2483–2494.
Zhao, P., et al., 2014. A nondestructive online method for monitoring the injection molding process by collecting and analyzing machine running data. Int. J. Adv. Manuf. Technol. 72 (5–8), 765–777.
Zhou, X., et al., 2017. Monitoring and dynamic control of quality stability for injection molding process. J. Mater. Process. Technol. 249, 358–366.