IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 2, 2019, accepted August 17, 2019, date of publication September 9, 2019, date of current version October 15, 2019.

*Digital Object Identifier 10.1109/ACCESS.2019.2940113*

# Node and Block-Based Development Tools for Distributed Systems With AI Applications

**MARCEL HAUCK[1], RÜDIGER MACHHAMER[2], LEVIN CZENKUSCH[2], KLAUS-UWE GOLLMER[2], AND GUIDO DARTMANN[2], (Senior Member, IEEE)**

[1]Research Group Business Informatics and Media Management, Mainz University of Applied Sciences, 55128 Mainz, Germany
[2]Environmental Campus Birkenfeld, Institute for Software Systems, Trier University of Applied Sciences, 55761 Birkenfeld, Germany

Corresponding author: Marcel Hauck (marcel.hauck@hs-mainz.de)

This project was funded by Federal Ministry of Food and Agriculture (BMEL) project IoT-Pilot (https://iot-pilot.umwelt-campus.de/) grant 2818LD003. Sourcecode is available at https://iot-pilot.umwelt-campus.de/software. Parts of this work are based on the master thesis of the first author. Special thanks to Anne-Kathrin Schirra and Peter Rock for the graphical abstract and the video, which was funded by Federal Ministry of Education and Research (BMBF) project COSY (https://cosy.umwelt-campus.de/) grant 01IS17073A.

**ABSTRACT** Internet of Things (IoT) and Artificial Intelligence (AI) are one of the most promising and disruptive areas of current research and development. However, these areas require deep knowledge in multiple disciplines such as sensors, protocols, embedded programming, distributed systems, statistics and algorithms. This broad knowledge is not easy to acquire and the software used to design these systems is becoming increasingly complex. Small and medium-sized enterprises therefore have problems in developing new business ideas. However, node- and block-based software tools have also been released and are freely available as open source toolboxes. In this paper, we present an overview of multiple node- and block-based software tools to develop IoT- and AI-based business ideas. We arrange these tools according their capabilities and further propose extension and combinations of tools to design a useful open-source library for small and medium-sized enterprises, that is easy to use and helps with rapid prototyping, enabling new business ideas to be developed using distributed computing.

**INDEX TERMS** Distributed computing, Internet of Things, machine learning, rapid prototyping, visual programming environments.

## I. INTRODUCTION

Today we are at the fusion of two disruptive technological developments: Internet of Things (IoT) and Artificial Intelligence (AI). IoT is the term for the connection of billions of machines world-wide, sensing an enormous amount of data. Typical applications are:

- Intelligent mobility where vehicles are connected and controlled by AI,
- Smart health, where data from millions of patients in smart hospitals is collected,
- Industrial IoT where all machines within the production are connected and synchronized and processes are optimized.

Without intensive data analysis this information would not generate new business ideas. Therefore, the integration of AI in the IoT is the consequent evolution of the next years. We give the reader a brief overview about the recent technological trends and the history of AI. Based on this overview, we present state-of-the-art software tools for IoT and AI and arrange them according to defined criteria. Small and medium-sized enterprises (SMEs) often do not have qualified programmers, IoT and AI specialists. In this paper we give a review of software tools that enable SMEs to rapidly develop new prototypes for IoT business ideas. As shown in Figure 1, we use a tool to program a successful generic IoT device *IoT Octopus* [1] based on the ESP8266.

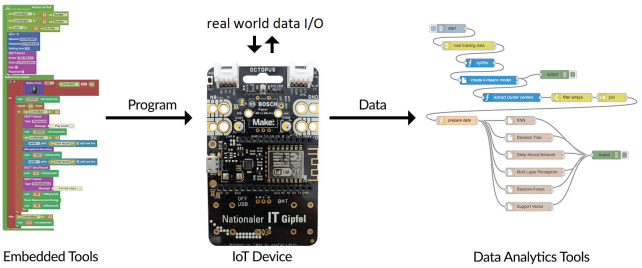



**FIGURE 1. Concept for IoT and AI tools (here with Ardublockly and Node-RED).**

Using IoT devices for new business ideas, programming tools for the sensing, communication and signal processing

The associate editor coordinating the review of this manuscript and approving it for publication was Zhen Ling.

must be developed (*Embedded tools* in Figure 1). With these tools individual IoT devices can be configured for specific applications. However, today IoT business ideas consist of multiple aspects, such as subscriber and publisher of information and data, cloud and databases and data analytics as well as AI for analysis and prediction of data. Therefore, we also need tools combining communication, data storage, cloud services, and data analytics. This is given by the right part of Figure 1 (*Data Analytic Tools*, here based on Node-RED from IBM). The represented flows organize distributed computing in Cyber-Physical Systems (CPS).

### A. INTERNET OF THINGS AND CYBER-PHYSICAL SYSTEMS

CPS describe the technical development for equipping common real objects with network-capable embedded systems [2]. These embedded systems collect data from the objects to send it via wifi or similar networks to IT applications or to deliver commands from the applications to affect the object. This development can be subdivided into 3 layers, the application, communication and perception layer [3]. Using the example of a temperature controlled fan,[1] the temperature sensor and the speed controller of the fan are implemented within the perceptual layer. The application layer uses the measured data to generate the control commands, which are transmitted via the communication layer. The IoT stands for the sum of the possible (interconnected) networks connecting the most diverse devices in all possible applications.

### B. ML IN IOT

Recent research indicates that IoT is very suitable for Machine Learning (ML) applications. ML requires a large amount of data to learn, which IoT/CPS can deliver. Instead of programming the desired temperature of 27 degrees for the fan from the above example, an ML approach would be to collect additional environmental data such as day, wind speed, outdoor temperature and the data to be considered from the user (from their personal devices) and encourage users to control the fan for the first few days according to their behavior. The algorithm begins to learning the habit of the person depending on the environmental data and starts suggesting the perfect needs. The learning process will take some time, but the prediction of the AI will more and more often guess the desired setting correct and thus exceed a static setting in the long run. Looking at current developments in the field of speech and video recognition, smart home applications already ease people's lives.

### C. SOFTWARE FOR ML/IOT

To keep attached to the different layers, for the application layer powerful python libraries like scikit-learn or TensorFlow exist, which provide the most common ML algorithms. IoT devices like Arduino or ESP8266/ESP32-based chips can be easily programmed with the Arduino IDE to perform the tasks of perception layer, and the basic network skills which fit the needs for an easy communication between the devices is quite easy, due to modern smartphone usage. Simple network knowledge of opening an access point or accessing a network is common knowledge. The development of applications similar to our fan, however, requires complex knowledge of IoT, ML and intensive programming skills.

### D. DISTRIBUTED COMPUTING

Latest research in the area of fog/edge computing leads to resource optimization by a smart distribution of tasks between participating devices, similarly, data flows have to be adapted for complex cloud applications. IBM's Node-RED offers possibilities to organize these tasks in an easy-to-understand way. Graphical elements organize data input and output, data manipulation methods or the use of ML algorithms with Python libraries. In addition, almost any program on any computer in the network can be controlled close to real time. Database operations or the use of web protocols such as Message Queuing Telemetry Transport (MQTT) or Hypertext Transfer (HTTP) are made available to a broad user community in an easy-to-use way.

### E. STRUCTURE

In this paper, we will give an introduction to basic ML concepts for IoT in Section II-A-II-C. Subsequently, in Section III, we introduce basic visual programming tools for IoT and AI and present a combination of a block-based programming tool for embedded programming as well as a node-based tool for IoT platform development and data analytics. Finally, in Section IV we present a case study in which the selected tools are applied in an IoT use case and give an outlook on further development and research.

## II. MACHINE LEARNING OVERVIEW

Today, AI is a branch of computer science. The goal is to enable machines to perform tasks *intelligently*. However, it is not defined what *intelligent* means in this context. First applications can be understood as expert systems. In the 1970s, systems were introduced that stored knowledge by a limited set of rules and were able to draw conclusions from this [4].

Rule-based systems can result in fast decisions, but require a high maintenance effort. The existing rule set must be regularly updated and extended. In addition, it is hard to provide solutions for previously unknown situations. In complex systems no new knowledge can be generated with it. Real AI systems, on the other hand, should allow to find decisions without a predefined set of rules, hence they should be able to learn tasks based on given data.

Whereas AI can be regarded as a generic term, ML describes the process of evaluating data with algorithms, learning from it, and finally, making decisions or predictions [5]. In general, the algorithms used in ML can be divided into three learning styles. If many data sets are available, that have already been marked with known solutions, it is practical

[1] which is a famous example to illustrate IoT, multiple tutorials are avalaible, e.g. https://dzone.com/articles/esp8266-wi-fi-fan-controller

to use supervised learning. In unsupervised systems, the raw data is sufficient to recognize patterns. In the third version, the machines use feedback from the environment to adapt their actions. This is often called enhanced or reinforcement learning. In the following, we will introduce the basic concepts and give a short summary for the use in IoT.

In ML, existing data (experience) is used to generate knowledge. In the area of classification, the available data can be assigned to classes. Examples are the differentiation of e-mails into spam and non-spam, or the decision whether a football player should be hired or not. Multiclass classifications such as customer groups are also possible.

Whereas a linear classification allows only discrete decisions (Yes or No, $-1$ or $+1$), a logistic regression allows the output of continuous values between 0 and 1. This corresponds to a probability which is a measure for the prediction of an event. If an optimal separation of the data is to be achieved, a support vector machine can be used. This maximizes the distance of the dividing line or plane to the two nearest data points of different classes.

A rather simple method of *Regression* is the *Linear Regression*. A weighting vector is optimized with the training data. This allows predictions to be made for previously unknown data points. Whereas the classification makes a strict separation, the output here is not limited to a value range. *neural networks* (NN) can also be used for both *classification* and *regression*. The foundations for this technique (machine neurons) were already laid in the 1940s. The system can be seen as a combination of many small single decision units. These so-called *Perceptrons* are linked by weighted edges. Within each unit, incoming signals are weighted and, after reaching a predefined threshold value, passed on to their successors as an output signal. For more details we refer to classical books on neural networks [6] and latest research [7].

### A. SUPERVISED LEARNING

Supervised learning (SL) divides the data into two parts (training set and test set). The first part is used to train a complex model, which afterwards can be tested with the second part. This model can then be applied to previously unknown data to make a decision or prediction. In many cases, the more data, the better the ML model, and IoT can produce a large amount of data and therefore, both terms fit together to train models for unknown and complex systems.

**Algorithm 1** KNN

**Input**: Data samples $D$; new sample $x$
**Output**: Class for sample $x$

Chose the $k$ nearest neighbors to $x$ from $D$
Identify the class, with the majority of points

The $k$-nearest neighbors algorithm (KNN) is often used because of its simplicity and resistance to remote outliers. For each data point in the test data, the $k$-nearest training data points are selected [1]. The new data point can then be assigned to the class that has the majority of these nearest points. The algorithm is generally described as follows:

### B. UNSUPERVISED LEARNING

Even if the existing training data is not labeled, patterns or groupings can still be found. These algorithms find clusters in the data. The data points contained should be very homogeneous among each other and very heterogeneous with respect to points of other groups. A representative of this group is the *K-Means* algorithm. First, $k$ random centers are chosen for the clusters and the nearest points are assigned to these centers. Then, for example, the mean value of the points is determined as the new position of the center. This reordering allows individual points to be attached to another cluster. The procedure is repeated until the midpoints are no longer shifted [8] or the classes of the data points are no longer changed [9]. Finally, the $k$ separated groups remain. The result depends, among other things, on the choice of the first centers. The functionality of the algorithm can be summarized as follows:

**Algorithm 2** K-Means

**Input**: Data set $D$; number of cluster $k$
**Output**: $k$ Cluster with assigned data points and $k$ centers

Chose $k$ random positions and define the centers
**repeat**
  Assign to every Point in $D$ the closest centre
  Calculate the new positions of the centres (e.g. mean of cluster samples)
**until** *centers are not moved*

### C. REINFORCEMENT LEARNING

The above methods can be used if a lot of training data is available from which knowledge can be generated. If there is an environment holding currently little data, but which can be simulated repeatedly, the use of reinforcement learning is practical. In this environment there is an intelligent program (agent). It is given a goal that it should achieve, for example maximizing the points in a game. It receives feedback about its success through rewards and punishments [10]. Over the years, many ML algorithms have emerged. It can be seen that they work better or worse depending on the application. In order to find the best approach, there are so-called *Cheat Sheets* which can help for an initial preselection. In Figure 2 the basic structures were summarized.

It is recommended to try out several approaches in parallel in order to achieve the best possible result. Experts with deep knowledge about the algorithms and programming languages are difficult to find for SMEs. Therefore, we want to present an overview of possible visual development tools for AI and also for the programming of IoT devices.

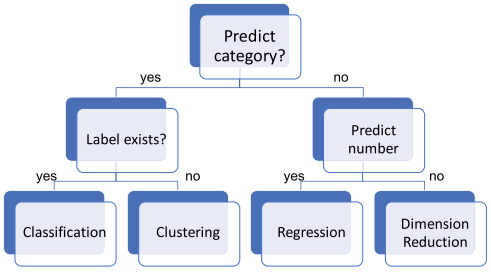



**FIGURE 2.** Decisions for different algorithms.

## III. VISUAL PROGRAMMING ENVIRONMENTS

After an overview of basic AI techniques was provided in the previous chapter, an introduction to visual development environments is given now. In this section, we target two different development aspects:

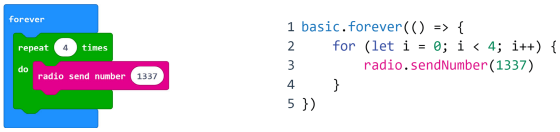
- IoT devices: Here we need tools for the programming of the embedded hardware for the sensing, signal processing and communication of IoT devices.
- Cloud and AI: Here we need tools to design the complete IoT-platform with databases (clouds) and decision making (AI).

The underlying concept of visual languages can be interpreted in two ways. On the one hand there are languages that process visual information. This applies, for example, to the representation of graphs [11]. On the other hand these concepts have a collection of visual programming elements [12]. Only languages corresponding to the latter description are relevant for this work.

Node based programming (NBP) describes a programming paradigm that brings data as modular objects into a node-based data flow. However, this does not have to be a visual programming that offers support to technical laypersons.

In the following only programming *environments* are examined, regardless of which visual programming *language* (for example JavaScript) they are realized in. The name VPE is used for this. They provide a framework in which visual (programming) elements can be lined up. A previously defined program code is then generated from the individual components. It is often also possible to enter variable values in form of input fields or selection lists. To set up and extend a VPE there is often a development environment in which, for example, new elements can be administered. The following elements must be defined for the end user to be able to use blocks:

- Graphic design (e.g. color of the block and contained images / pictograms)
- Description texts (both labels and help texts or further links)
- Interaction possibilities (Which values – e.g. numbers or texts – may be entered? Which interaction possibilities – e.g. selection lists – are there?)




(a) Graphic programming. (b) Corresponding code

**FIGURE 3.** Generation of Code from a graphic programming block.[2]


(a) ArduBlock[3]. (b) Scratch[4].

**FIGURE 4.** Comparison of Ardublock and Scratch.

- Interfaces (Are there input or output possibilities to other blocks?)
- Generated code (Which program code should be generated dynamically?)

Kurihara et al. state three reasons for the use of VPE [13]. First, no syntax errors can occur because no code manipulation is necessary. Second, the programming environment can be described in natural language so that it is easy for the user to understand. Third, even complex functions with case distinctions and dependencies can be integrated into simple building blocks. This prevents necessary tasks, such as type conversions, from not being used.

An example for the graphical programming of the single board computer *micro:bit* using the VPE *Microsoft MakeCode* is represented in Figure 3.

The code on the right is automatically generated from the block on the left. This example shows the reduction of complexity, as the user does not need to know how to address the radio unit of the device.

### A. CATEGORIZATION

In general, there are two categories in which VPE can be classified: block-based programming (BBP) and node-based programming (NBP). BBP refers to environments in which graphical blocks are connected via matching recesses, such as in a puzzle. Especially for young users the Massachusetts Institute of Technology (MIT) developed *Scratch*. In Figure 4b a short example application is shown. From the selected blocks a program code usable by the computer is automatically generated in the background. The user only has to take care of the semantic correctness of the application. The displayed program can be used to move a figure, change its color or output a sound. *ArduBlock* is a second VPE

[2]https://makecode.microbit.org/
[3]https://www.umwelt-campus.de/iot-werkstatt/
[4]https://scratch.mit.edu/


(a) BPMN-Process[5]. (b) IBM Node-RED-Process.

**FIGURE 5. Comparison of BPMN with NBP.**

for programming microcontrollers. Based on the Arduino platform, sensors, actuators or displays can be controlled. Figure 4a shows a simple traffic light circuit.

The aforementioned development areas are characterized by the use of case distinctions (`IF-THEN-ELSE`) and loops (`WHILE`). In companies, application steps are often modeled as processes. These can be represented by business process model and notation (BPMN), for example. As visible in Figure 5a, process steps are connected with lines, which results in a process. This pattern is also used by NBP environments. Via connections, data can be exchanged between the individual nodes. An example of this is the process shown in Figure 5b. The application *Node-RED* of the company IBM offers function modules (e.g. MQTT connectors), which are suitable for development in the IoT area. In the following we compare existing frameworks based on their capabilities.

A total of 29 VPEs were determined in the area of the BBP (Table 1). The presented VPEs are partly based on each other. In Figure 6, the development is shown based on *Scratch*.

In NBP the overview looks a bit different. Of the 17 VPEs determined, only three can be called via the browser (*Flow*, *Flowhub IDE*, and *Node-RED*). Most of the applications also have a proprietary license (e.g. *RapidMiner Studio*). *Enrect* can be seen as a special case. This system uses both blocks and connections between them. It is available in Japanese, Estonian, and English and focuses on continuing education with large icons.

### B. VPE TOOLS FOR SMES

Low-coding platforms are very easy to handle, but can usually only be used in a special case (e.g. image recognition). Text-based coding (e.g. with a code editor) allows a very high flexibility. However, computer science knowledge is required to write code in a programming language. VPEs are a mixture of both worlds. Due to their ready-made building blocks they are quite easy to use. Moreover, they are not bound to a specific purpose. Therefore, they are a perfect tool for SMEs, to develop new business ideas. Today, the early development of new prototypes is very important for the success of a company. Therefore, in this section we want to present a combination of VPEs that allow this rapid prototyping approach and can lead to an acceleration of new product ideas in IoT and AI.

Both BBP and NBP environments are suitable for application in a SME. Only BBP can be applied for programming the IoT kit utilized in the use case, since it has `IF-THEN-ELSE`-based programming. However, they are not suitable for further processing of the data using ML algorithms. Therefore one BBP- and one NBP-VPE are selected below. Table 1 contains open source applications as well as proprietary applications. Table 2 gives an overview of the selection criteria used. Finally, two BBP candidates meet all of our requirements:

The *BlocklyDuinoReboot* programming blocks can be translated into Arduino code as well as into other programming languages (e.g. `PHP` or `Lua`). The respective code can be called up via the tabs at the top of the screen. The system is not responsive, i.e. it does not adapt to the size of the display device. In addition, there is no component integrated that can be used to communicate with the Arduino environment. This must be integrated separately, for example from *BlocklyDuino*.

*Ardublockly* on the other hand offers a responsive web design. It can also be used to implement desktop applications which run without an additional browser. In contrast to the aforementioned environment, the program code generated by the environment is displayed here. In Figure 7 this can be seen on the right side of the screen. Changes to the code are indicated by a coloured marking. Users can see the effects of block changes directly. In addition, more colored elements (e.g. in the header area) are used in the design of the entire interface.

The tool contains a server application developed in `Python` that can communicate with the Arduino environment. For example it sends the generated Arduino code to the Command Line Interface (CLI) `arduino_debug.exe`, which compiles and uploads the machine code to the microprocessor. The JavaScript (JS) files used in the frontend can optionally be compressed using the *Closure Compiler*.[6] They are sent to a Google server and sent back optimized. The environment is already used productively (e.g. by *Kniwwelino* or *Oxocard*) and can therefore be considered as stable. Due to the clear advantages of *Ardublockly* it is chosen as the BBP environment.

The decision for a NBP can be made quickly. Table 1 contains only five environments with open source licensing: *KNIME*, *Luna Studio*, *Orange*, and *RAPTOR* with GNU GPL and *Node-RED* with Apache 2.0. Of these remaining environments, only *Node-RED* was developed with web technologies (e.g. JS) and can be used independently of the end device. It corresponds to the requirements of the rapid prototyping for business ideas in IoT and AI, since it was developed especially for the use in the IoT area. In the freely accessible library[7] it is possible to download over 1.000 predefined processes and more than 1.700 extensions. There are also additional packages included that provide ML nodes. This makes it possible to provide AI algorithms as simple flow elements.

[5]https://docs.camunda.org/manual/7.6/reference/bpmn20/events/bpmn/event-conditional.svg

[6]https://developers.google.com/closure/compiler/

[7]https://flows.nodered.org/

**TABLE 1.** VPEs, BSD: Berkeley software distribution.

| Type | Name | Editor | developed in | based on |
|---|---|---|---|---|
| BBP | Alice | Carnegie Mellon University | Java | |
| | ArduBlock | Arduino.CC | Java | |
| | Ardublockly | carlosperate | JS, Python | Blockly |
| | Arduino Create | Arduino | JS | |
| | Blockly | Google | JS | |
| | BlocklyDuino Enhanced | Kassah | JS | Blockly |
| | BlocklyDuinoReboot | *Collaborative* | JS | Blockly |
| | BlockPy | *Collaborative* | JS | Blockly |
| | Calliope mini | Calliope | JS | |
| | Grape | qfix robotics | *unknown* | |
| | GraspIO | Grasp IO | *unknown* | |
| | Kniwwelino | LIST | *unknown* | Ardublockly |
| | MakeCode | Microsoft | TypeScript | Blockly |
| | mBlock | Makeblock | Actionscript | Scratch |
| | mBlock Blockly | Makeblock | *unknown* | Blockly |
| | miniBloq | *Collaborative* | Python | |
| | Modkit | Modkit | Python | |
| | NETLab Toolkit | Van Allen | JS | |
| | NetsBlox | Vanderbilt University | JS | Snap! |
| | Open Roberta | Fraunhofer IAIS | C, JS | Blockly |
| | Oxocard | OXON | JS | Ardublockly |
| | S4A | Citilab | Scratch | Scratch |
| | Scratch | MIT | JS | |
| | ScratchJr | *Collaborative* | JS, Java, Objective-C | Scratch |
| | Snap! | University of California | JS | Scratch |
| | Snap4Arduino | Romagosa and GuillÃl'n | JS | Snap! |
| | Squeak | Kay | C | |
| | ToonTalk | Kahn | JS | |
| | Wyliodrin STUDIO | Wyliodrin | JS | Blockly |
| NBP | Embrio | Embrio | *unknown* | |
| | Enrect | Suzuki | *unknown* | |
| | Flow | AT&T | *unknown* | |
| | Flowcode | Matrix TSL | Visual C++ | |
| | Flowgorithm | Cook | C# | |
| | Flowhub IDE | Flowhub | JS | |
| | iCon-L | ProSign | *unknown* | |
| | KNIME | KNIME | Java | |
| | LabVIEW | National Instruments Corporation | *unknown* | |
| | Luna Studio | Luna | Haskell | |
| | Node-RED | IBM | JS | |
| | Noodl | Topplab | *unknown* | |
| | Orange | University of Ljubljana | Python | |
| | RapidMiner Studio | RapidMiner | *unknown* | |
| | RAPTOR | Martin Carlisle | *unknown* | |
| | Simulink | MathWorks | *unknown* | |
| | Visual Logic | PGS Systems | *unknown* | |

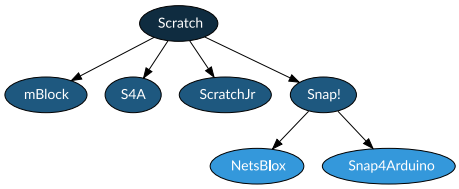



**FIGURE 6.** Further development of a VPE using the example of scratch.

**TABLE 2.** BBPs.

| Name | Open Source | Web-based | Generated Code visual | Code gen. for Arduino | Arduino lib. usable | support of Microcontr. | Offline | Project start |
|---|---|---|---|---|---|---|---|---|
| Ardublockly | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 22.04.2012 |
| BlocklyDuinoReboot | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 27.10.2013 |
| Oxocard | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | 06.07.2017 |
| Open Roberta | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | 02.03.2014 |
| Blockly | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | 27.10.2013 |
| MakeCode | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | 24.01.2016 |
| Snap4Arduino | ✓ | ✓ | ✗ | ✓ | ? | ? | ✓ | 31.01.2016 |
| Scratch | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | 11.09.2016 |
| ScratchJr | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | 03.01.2016 |
| ToonTalk | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | 02.02.2014 |
| BlockPy | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | 27.10.2013 |
| NetsBlox | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | 10.03.2013 |
| Snap! | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | 10.03.2013 |
| ArduBlock | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | 06.02.2011 |
| miniBloq | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | 25.08.2013 |
| mBlock | ✓ | ✗ | ✓ | ✓ | ✗ | ✗ | ✓ | 08.03.2015 |
| S4A | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | 08.10.2013 |
| Squeak | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ? |

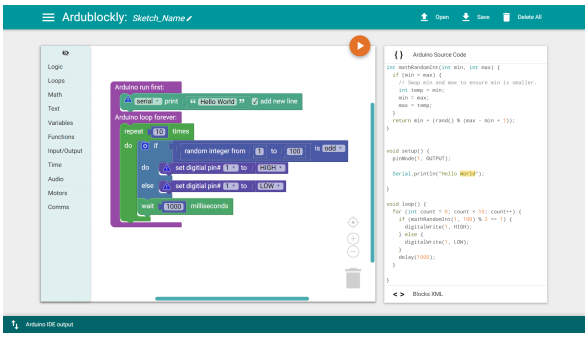



**FIGURE 7.** Sample application in Ardublockly.

## IV. COMBINATION TOOL AND IOT CASE STUDY

The previously selected tools *Ardublockly* and *Node-RED* are combined to a tool chain uploaded to https://iot-pilot.umwelt-campus.de/software. We extend this tool chain with super blocks of the most important sensors and actuators in order to offer an intuitive sensor connection for almost any physical quantity, such as distance, temperature or luminous intensity. We are constantly expanding these blocks with functions such as providing an access point, a LoRa interface or control of actuators, but the goal is to form a broad community that participates in the further development of the system. The generation of a new IoT idea can now take place via an interface as follows: First, the user can configure the IoT device in *Ardublockly* and select multiple sensors, actuators, and communication protocols such as MQTT. In a second step, the user can export the MQTT-configuration to *Node-RED* via a cut & paste-mechanism. Finally, the user can set-up the entire platform by *Node-RED* including multiple data analytic tools provided by common ML-frameworks.

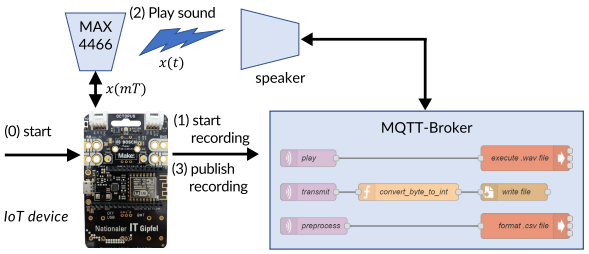



**FIGURE 8.** Data generation [1].

### A. SOUND-BASED LOCALIZATION

The following section describes how to use the combination tool in the use case *IoT-based sound localization*. It is based on the algorithms and procedures developed by Dziubany *et al.* [1]. The idea is localize the position of an object in space using a single, inexpensive microphone and a loudspeaker. The speaker always remains unchanged in the same place. To do this, a part of a tabletop is parceled into 16 square, equally sized areas, which are assigned a consecutive number. In each of the sections, five sound measurements – a total of 80 – are taken, which are processed and saved as a file (training data). Three representatives per square are determined using the *K-Means* algorithm. An IoT kit is used to record the data. Figure 8 shows the distributed process of data generation. The IoT kit triggers the sound output via Node-RED and MQTT using the topic *play*. Due to the limited resources, the raw data is collected as bytes, which are then *transmitted* in 20 MQTT packages (limited package size) to the Node-RED device to first compute integer values. These values are normalized using MATLAB for further processing in the ML nodes.

For the training phase described above, a sequence in *Ardublockly* was created. Figure 10, shows the finished configuration. Besides general blocks (control structures, MQTT, hardware button) three individual elements (*microphone recording*, *send recording via MQTT*, and *reset measured value memory*) can be seen, which were implemented for this use case. Due to the simple creation with the BlocklyFactory[8] the integration takes only a few minutes. Thus, the process of training data recording is covered and can be easily varied in the VPE.

In Figure 9, the generated Node-RED flow for the evaluation of the training data is mapped using ML algorithms.

[8]https://blockly-demo.appspot.com/static/demos/blockfactory/index.html

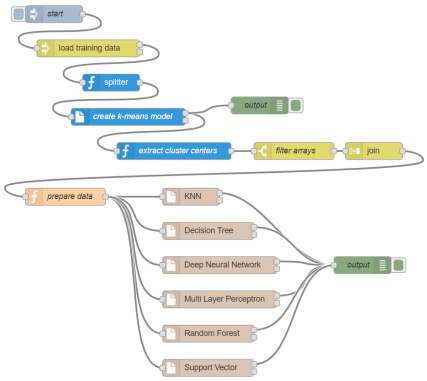



**FIGURE 9.** Training of the localization in Node-RED.

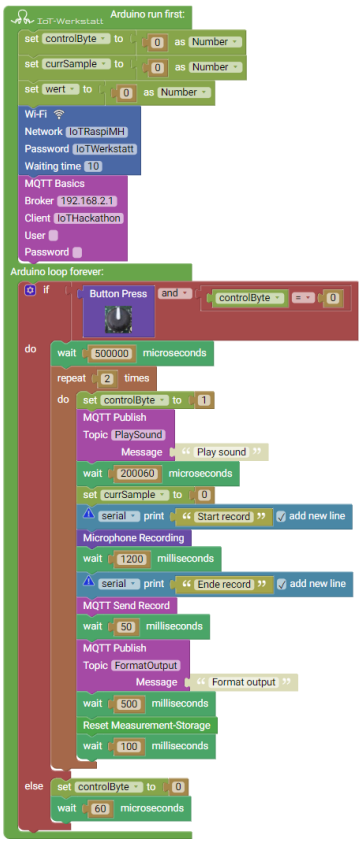



**FIGURE 10.** Training phase Ardublockly.

First the training file is read in there. The *Splitter* node groups five data samples and sends them to the *K-Means* node. There, three representatives are generated in each case. The *Extractor* node reads the attribute `cluster_centers_` and sends it to the following *join* node, which unites all representatives in an array.[9]

The function node *Prepare Data* adds labels to the training data, since they are required for the use of a SL procedure. Since the Node-RED library `contrib-machine-learning` contains several SL algorithms, they can be tested very easily in parallel without programming

[9]Information on the available attributes at https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

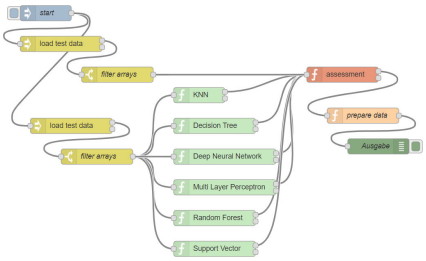



**FIGURE 11.** Accuracy determination with test data in Node-RED.

knowledge. The library accesses ML functions from pythons *scikit-learn* and *TensorFlow*. Figure 9 shows the whole process with the creation of six ML learning algorithms.

The sound localization is to predict in which spatial area of the table they were recorded for previously unknown measured values. If the previously created 16 *K-Means* models were individually checked, the process would take a very long time and produce 16 different results. Therefore the SL algorithm *KNN* based on the *K-Means* representatives was chosen as the learning method. It can be used to quickly figure out which label the nearest neighbors have. This can then be assigned to the new measured value.

In addition to the 80 training data samples, a further 12 test values (in total 192) are available for each of the 16 areas.[10] They can be used to estimate how well the previously created models can predict new values. As can be seen in Figure 11, the training data is read *twice* into *Node-RED*. The upper node only reads the previously logged correct results for later comparison with the forecast. The lower node loads the actual test data in order to classify them according to different ML algorithms and thus check their suitability for the application. The corresponding *Ardublockly* blocks for the test phase is given in Figure 12.

Table 3 shows the results of a test run. Please note that the respective settings have not been optimized. The achieved values could be improved by further configurations. *KNN* provides the best result, which is why it can be considered as useful for a IoT-based sound localization. In addition, the application example is kept very simple. In a more complex application, different devices could process the individual algorithms in order to transmit their results to a central unit.

As previously described, the system can be used to evaluate previously unknown measurements using the model created in the training phase. A new *Ardublockly* and *Node-RED* sequence has been set up to present the results even better to the user.

First, a new measured value is recorded with Ardublockly. The sketch required for this can be found in Figure 12. An MQTT node (*MQTT Send Record*) is integrated, which sends the recorded sound values to an MQTT broker (*MQTT Basics*). The return from the *KNN* model is output on the LED matrix of the IoT kit. To receive the values in Node-RED,

[10]https://github.com/dziubany/ML-Localization/tree/master/dataPaper

**TABLE 3.** Accuracy of ML algorithms. Own representation.

| ML-Algorithm | Accuracy | Settings |
|---|---|---|
| KNN | 87,5% | `[n=1; weights=distance]` |
| Multilayer Perceptron | 76,0% | `[max_iterations=1000; activation=relu; solver=gini; layers=10,10,10]` |
| Support Vector Machine | 70.8% | `[kernel=rbf; c_penalty=1]` |

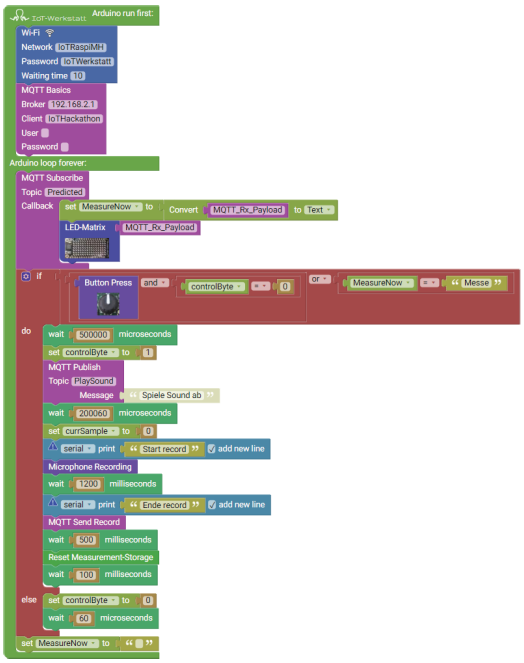



**FIGURE 12.** Testing phase Ardublockly.

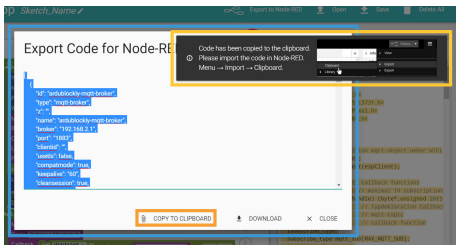



**FIGURE 13.** Export Node-RED settings from Ardublockly.

the settings for the *MQTT connection* and the *Topic* would normally have to be transmitted manually.

To simplify the process and prevent transmission errors, a new *Node-RED Export* button is implemented in Ardublockly. A code of the nodes used for Node-RED is generated from the information read with it. This can easily be imported via the clipboard (*Menu* → *Import* → *Clipboard*). Figure 13 shows a generated code (frame), the clipboard button (frame) and the hint to paste (frame) in Node-RED.

In Figure 14, the flow for processing the incoming signal data and evaluation is shown in *Node-RED*. In the individual function blocks (*ByteToInt*, *prepare data* and *ArrayToString*) the information received from the microphone is processed. This processing is especially for the method of sound localization and therefore does not have to be included in the

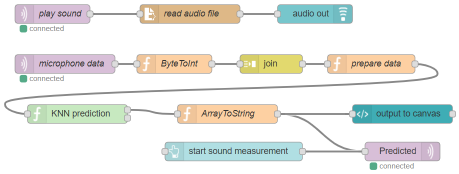



**FIGURE 14.** Node-RED configuration for sound locating predictions.

library as generalized *Node-RED* nodes. The value predicted from the *KNN* model is not only displayed on the LED matrix of the IoT kit, but also as text in a *Node-RED* dashboard. For this the *Node-RED* library `node-red-dashboard`[11] is used.

This offers the possibility for the user to operate and evaluate the *Node-RED* process without knowing the program code or the *Node-RED* flows. It is therefore an additional level of abstraction that allows it to be used at an even higher level. Here it is also possible, for example, to start the sound measurement by means of buttons or to display received measured values directly as a graph. However, the display formats are quite limited and, for example, in the case of two-dimensional diagrams as Y-axis, a time specification must be used.

A free plot, as it is possible with individual configurations in the Python library *Matplotlib*[12] is therefore not realizable.

The use case presented in this section shows that the selected development tools can map the entire process from data generation, data preparation and data processing to user-friendly data representation.

### B. USER INTEGRATION

The combination of the tools will be made publicly available on the website of research project IoT-Pilot (*https://iot-pilot.umwelt-campus.de/*). SMEs and interested private people are instructed in tutorials to use them for their own purposes. We are looking forward to

- broad interest and participation in crowd applications for smart city environments on the part of private users.
- rapid prototyping attempts of SMEs to test their new business ideas.

In the context of an IoT maker sphere SMEs can advertise projects to a broad community of IoT-experts, students or laymen interested in CPS, IoT or ML. Laymen should as well be animated to publish their own projects. The mutual remuneration of the project partners can be handled via crypto technologies using smart contracts in order to enable fair

[11] https://flows.nodered.org/node/node-red-dashboard
[12] https://matplotlib.org/

compensation in micropayment e.g. for crowd sensing, or to compensate copyrights between idea developers and realizing SME. Distributed cloud computing systems allow data which is delivered by broad user communities in order to gain insights from this data that will be of value to these communities. Distributed cloud computing system enables data to be delivered from a broad user base to evaluate them and gain insights which adds value for these communities.

## V. CONCLUSION

The aim of this paper was to evaluate and further develop existing development tools that can be used in IoT- and AI-based business models in SMEs. The theoretical basics served as a basis for the practical implementation. The goal could be achieved by a combination of several VPE. In order to enable as many developments as possible, the system should be universally applicable and have a modular structure.

BBP environments are suitable for the programming of microcontrollers. They offer the advantage that they can be easily translated into program code. It must be possible to use new sensors (e.g. microphone) and boards (e.g. ESP32). For this purpose the libraries published by the hardware manufacturers must also be able to be integrated. The application should be web-based and responsive so that it can also be accessed on different devices. Only *Ardublockly* meets the aforementioned requirements and was therefore selected as BBP. After enrichment with our super blocks, it meets all our application requirements. However, it turned out to be disadvantageous that it is not well suited for process-based data processing and preparation.

In this area, NBP environments can play out their advantages. Although they are not suitable for programming microcontrollers, they can link the collected information with available company data and evaluate it using AI procedures. The node-based representation facilitates the use, since no own program code must be developed. Furthermore, different approaches (e.g. NN vs. KNN) can be compared quickly.

The web-based NBP *Node-RED* was selected to avoid media discontinuity when switching to a classical application program. Thanks to its open source licensing and many available extensions, it is well suited for the use in IoT-based enterprises. As shown in this paper, targeted adaptations are useful, for example to be able to use additional functionalities such as new ML algorithms.

As the use case *IoT sound localization* shows, the system offers tools that cover the entire Data Analytics tool chain from data collection to data presentation using distributed systems.

The selected systems are not tailored to a specific field of application yet. Thanks to their ease of use and universal expandability, even new business models can emerge. Medium-sized companies can digitize analog processes and even use complex learning processes from the field of AI to further improve their products or to develop new ones. More experienced users can distribute complex tasks to different devices or have the distribution performed automatically using ML.

## ACKNOWLEDGMENT

This project was funded by Federal Ministry of Food and Agriculture (BMEL) project IoT-Pilot (https://iot-pilot.umwelt-campus.de/) grant 2818LD003. Sourcecode is available at https://iot-pilot.umwelt-campus.de/software. Parts of this work are based on the master thesis of the first author. Special thanks to Anne-Kathrin Schirra and Peter Rock for the graphical abstract and the video, which was funded by Federal Ministry of Education and Research (BMBF) project COSY (https://cosy.umwelt-campus.de/) grant 01IS17073A.

## REFERENCES

[1] M. Dziubany, R. Machhamer, H. Laux, A. Schmeink, K.-U. Gollmer, G. Burger, and G. Dartmann, "Machine learning based indoor localization using a representative k-nearest-neighbor classifier on a low-cost IoT-hardware," in *Proc. 26th Eur. Signal Process. Conf. (EUSIPCO)*, Sep. 2018, pp. 2050–2054.

[2] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1125–1142, Oct. 2017.

[3] H. Xu, W. Yu, D. Griffith, and N. Golmie, "A survey on industrial Internet of Things: A cyber-physical systems perspective," *IEEE Access*, vol. 6, pp. 78238–78259, 2018.

[4] S. J. Russell, P. Norvig, and E. Davis, *Artificial Intelligence: A Modern Approach* (Artificial Intelligence), 3rd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2010.

[5] M. Copeland. (Jul. 2016). *The Difference Between AI, Machine Learning, and Deep Learning|NVIDIA Blog*. [Online]. Available: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[7] W. G. Hatcher and W. Yu, "A survey of deep learning: Platforms, applications and emerging research trends," *IEEE Access*, vol. 6, pp. 24411–24432, 2018.

[8] S. Marsland, *Machine Learning: An Algorithmic Perspective* (Machine Learning Pattern Recognition). Boca Raton, FL, USA: CRC Press, 2009.

[9] D. Singh and C. K. Reddy, "A survey on platforms for big data analytics," *J. Big Data*, vol. 2, no. 1, p. 8, Dec. 2015.

[10] A. Nandy and M. Biswas, "Reinforcement learning basics," in *Reinforcement Learning*. Berkeley, CA, USA: Apress, 2018, pp. 1–18.

[11] K. Zhang, *Visual Languages and Applications*. New York, NY, USA: Springer, 2007.

[12] G. De Luca, Z. Li, S. Mian, and Y. Chen, "Visual programming language environment for different IoT and robotics platforms in computer science education," *CAAI Trans. Intell. Technol.*, vol. 3, no. 2, pp. 119–130, Jun. 2018.

[13] A. Kurihara, A. Sasaki, K. Wakita, and H. Hosobe, "A programming environment for visual block-based domain-specific languages," *Procedia Comput. Sci.*, vol. 62, pp. 287–296, Jan. 2015.

**MARCEL HAUCK** received the B.Sc. degree in media, IT, and management from the Mainz University of Applied Sciences and the M.Sc. degree in media informatics from the Environmental Campus Birkenfeld, Trier University of Applied Science, in 2018. He is currently pursuing the Ph.D. degree with the Research Group Business Informatics and Media Management, Mainz University of Applied Sciences, and also with the Johannes Gutenberg University of Mainz.

**RÜDIGER MACHHAMER** received the master's degree in computer science from the Trier University of Applied Sciences, Environmental Campus Birkenfeld, Germany. He is currently pursuing the Ph.D. degree with the Research Group Distributed Systems at the Institute for Software Systems (ISS). His research interests are in Machine Learning, Online Learning and Internet of Things.

**LEVIN CZENKUSCH** received the B.Sc. degree in computer sciences from the Trier University of Applied Science, Environmental Campus Birkenfeld, in 2018. He is currently pursuing the M.Sc. degree while working with the Research Group Distributed Systems at the Institute for Software Systems (ISS).

**KLAUS-UWE GOLLMER** received the Diploma in Biomedical Engineering from University of Applied Sciences in Hamburg in 1987 and the Diploma in Electrical Engineering from Technical University Hamburg-Harburg in 1991. He received the Ph. D. from University Hannover in 1996. Since 1999 he is Professor for Modelling and Simulation at Trier University of Applied Science. He is a member of the IoT Expert Group of the German National Digital-Summit. His major research interests include Internet of Things and Machine Learning.

**GUIDO DARTMANN** received Diploma and Ph.D. degrees from RWTH Aachen University, in 2007 and 2013, respectively. Since 2016, he has been a Professor of distributed systems with the Trier University of Applied Science. He is a member of the IoT Expert Group of the German National Digital Summit and a Founding Member of the IEEE Special Interest Group on Big Data Intelligent Networking. His current research interests include distributed systems, hardware software co-design, and machine learning.