Kölner Beiträge zur technischen Informatik
Cologne Contributions to Computer Engineering
Band 2/2022

Ergebnisse der Workshops 2022 des Forschungsschwerpunkts Vernetzte intelligente Infrastrukturen und mobile Systeme (VIMS)

Technology
Arts Sciences
TH Köln

Impressum:

Forschungsschwerpunkt Vernetzte intelligente Infrastrukturen und mobile Systeme
Technische Hochschule Köln
Fakultät für Informations-, Medien- und Elektrotechnik
Betzdorfer Str. 2
D-50679 Köln
tobias.krawutschke@th-koeln.de
Stand: Dezember 2022
ISSN: 2193-570X

Inhaltsverzeichnis

Editorial

Mit diesem, zweiten Band im Jahr 2022 wird die Veröffentlichung in der Reihe *Kölner Beiträge zur technischen Informatik* fortgesetzt. Die Fakultät für Informations-, Medien- und Elektrotechnik am Institut für Nachrichtentechnik ermöglicht Master Studierenden nicht nur aus dem Bereich der technischen Informatik eine Möglichkeit Ihre Forschung zu veröffentlichen, die im Rahmen von Forschungs- und Entwicklungsprojekten an der TH Köln und/oder bei Projektpartnern entstand.

Ziel ist es, die Ergebnisse laufender Arbeiten aus den Forschungs- und Entwicklungsaktivitäten des Forschungsschwerpunkts nach außen zu kommunizieren und Informatiker und Informationstechniker außerhalb des Forschungsschwerpunkts z.B. aus dem Kölner Raum einzuladen, neue Ergebnisse aus Wissenschaft und technischer Anwendung im Rahmen von *Cologne Open Science* zu publizieren.

In zwei Workshops wurden die hier veröffentlichten Themen diskutiert. Am 10.6.2022 wurden *Multi-Agent Reinforcement Learning for smart Computing Resource Allocation in the Industry 4.0 (Michael Urlaub, Julia Rosenberger)* und *Investigations on self-optimizing PID controllers based on neural networks and Implementation in a process control system (André Wittling)* vorgestellt. Am 25.11.2022 wurden *Visual detection of a charging station and implementation of a docking routine performed by an autonomous vehicle (David Kliewe)* und *Development of a Robust Peak Detection Algorithm in Time Series Data (Shezan Hossain Mahmud, Ender Akcöltekin, Dr. Cyrano Bergmann)* vorgestellt.

Der Herausgeberkreis freut sich, diesen neuen Band der Reihe der Fachöffentlichkeit zur kritischen Prüfung und zur möglichen Mitwirkung vorlegen zu können.

Rainer Bartz
Andreas Behrend
Andreas Grebe
Tobias Krawutschke
Hans W. Nissen
Beate Rhein
René Wörzberger
Chunrong Yuan

Multi-Agent Reinforcement Learning for smart Computing Resource Allocation in the Industry 4.0

Michael Urlaub
Technische Informatik (Master)
TH Köln
Köln, Germany
michael_martin.urlaub@smail.th-koeln.de

Julia Rosenberger
Automation and Electrification Solutions
Bosch Rexroth AG
Lohr am Main, Germany
julia.rosenberger@boschrexroth.de

***Abstract*—The fourth industrial revolution, also called Industry 4.0, describes the digitization process in industry. Data and information are of high significance as the number of data-driven business models increases rapidly. To handle the challenging requirements in data processing, e.g. real-time processing, data security and economic aspects, edge computing is increasingly favoured compared to cloud computing. The most limiting factor for edge computing is the resource limitations of the industrial edge devices. This study describes an approach to overcome these limitations using multi-agent reinforcement learning for allocation of computing resources in the Industrial Internet of Things to enable edge computing. The focus of the study lays on the experimental evaluation of the proposed concept and comparison of different hyperparameter configurations for the multi-agent-system.**

***Index Terms*—Reinforcement Learning, Multi-Agent-System, Resource Allocation, Industry 4.0, Industrial Internet of Things**

I. Introduction and Problem Statement

It is to observe that digitization processes in Industry 4.0 lead to a steady increase in networked devices, resulting in a high volume of data in the coming years. It is very likely that the development of recent years will be significantly exceeded. Researchers predict that the number of connected devices will increase at an annual growth rate of 12-17% from about 27 billion in 2017 to over 125 billion devices in 2030 [1].
The transfer of data to central cloud architectures is increasingly problematic in this context. The enormous amounts of data generated in the Internet of Things (IoT) would require the expansion of the communication infrastructure if transferred to the cloud [2]. IoT objects typically have limited resources, making direct interaction with the cloud infeasible. In addition, modern industrial and IoT infrastructures usually require very low latency [3], [4]. Besides, the speed, security of data and data transmission as well as the high demand for costly bandwidth are challenges that have led to the rethinking of the centralized architecture. Decentralization in the sense of an intelligent edge computing system represents one solution approach [5].
This enables the minimization of latency, the reduction of data security risks and the improvement regarding bandwidth bottlenecks. However, since edge devices have little computing power compared to the cloud architecture, intelligent resource management on the edge devices is necessary. In this work, we want to enable edge computing by optimal usage of the resources of the edge devices. For industry, we define edge devices as the devices directly involved in the production process. We consider devices assigned to the field level and control level of the automation pyramid. Thus, edge devices can range from smart sensors up to powerful industrial PCs.

In this work, a representative edge device of medium performance is considered, namely the industrial control unit ctrlX CORE from Bosch Rexroth with a 64 bit quad core ARM CPU, 1 GB RAM and 4 GB eMMC memory. The resource overhead by the multi agents has to remain low to ensure that the agents can be deployed on a variety of edge devices.

In this study, a new approach for decentralized resource allocation for the Industrial Internet of Things (IIoT) is evaluated. The content presented in this study is the result of a master thesis at the TH Köln in cooperation with Bosch Rexroth AG. The very basic idea of allocating computational resources through multiple agents and reinforcement learning is described in [6]. This work differentiates from the first sketch [6] in particular in the design of both action and state space, and the consideration of dynamic network changes in addition to the varying number of processing algorithms and data streams. Further developments of the resource allocation system are described in [7].

II. Related Works

Agent-systems are of increasing relevance in industry [8], [9]. Additionally, DRL is increasingly applied for resource allocation in various fields like transportation [10], job shop scheduling [11] and network resources [12] and is said to be an important technology for the Industry 4.0 [13].

The field of computing resource allocation in resource-limited networks is studied several times in the context of mobile edge computing, e.g. in [13], [14]. Most existing approaches are single-agents or centralized Multi-Agent System (MAS) [14], [15]. As the subject of this study is a fully decentralized approach, the approaches [16]–[19] are to be delineated. The most common approach is task offloading from the limited edge devices to a server as described e.g. in [13], [14], [16], [17]. In contrast, our approach pursues the goal to optimally use the existing resources of the high number of network participants instead of requiring an additional

server. Another fully decentralized Multi-Agent Reinforcement Learning (MARL) approache is proposed in [19], but its optimization objective is load balancing for controller in software-defined networks. In [18], an approach focusing on resource allocation is presented. Considering both computing and communication resources, the approach is of higher complexity than our approach. The Reinforcement Learning (RL) agents decide about the need for task offloading and take into account the communication resources, size of the input data and computational effort for executing the task. As our main objective for edge computing is the processing of streaming data, it is to assume that the processing task never ends. Thus, our approach differs from classical task offloading. A promising DRL approach that also focuses on distributed stream data processing is proposed by Li et al. [20]. It differs from our work mainly in its single-agent architecture.

So far, no work on intelligent, fully decentralized methods for resource allocation for streaming data processing tasks in industrial environment is known.

III. Background

A. Reinforcement Learning

In RL that is based on the trial-and-error principle, an agent interacts with an environment and collects experience values. These experiences are represented in the form of rewards, which evaluate the last action with respect to the goal achievement. With the help of the rewards a so-called policy can be trained, so that future actions are more goal-oriented. Mathematically, RL can be described with the Markov Decision Process (MDP). The tuple (S, A, P, R, γ) defines a MDP [21].

- S : set of states, $s \in S$
- A : set of actions, $a \in A$
- P : transition function $P : S \times A \times S \to [0, 1]$
- R : reward function $R : S \times A \times S \to \mathbb{R}$
- γ : discount factor, $\gamma \in [0, 1)$

B. Multi-Agent Reinforcement Learning

If several independent agents interact in a common environment, it is called a MAS. Depending on the goal, the agents compete or cooperate [22]. To define the optimal strategy for a single agent in advance is very difficult or even impossible, therefore self-learning methods are helpful.
RL is a Machine Learning (ML) concept that allows the agents to find their own optimal policy [23]. In this case we speak of a MARL, which can be defined as Stochastic Game (SG) [24] or Markov Game (MG) [25].
The tuple $(N, S, \{A^i\}_{i\in N}, P, \{R^i\}_{i\in N}, \gamma)$ defines a MG [21]:

- N : set of agents, $N = \{1, ..., n\}$ and $n > 1$
- S : set of states, $s \in S$
- A^i : set of actions of agent i, $A := A^i \times ... \times A^n$ and $a \in A$
- P : transition function, $P : S \times A \times S \to [0, 1]$
- R^i : reward function of agent i, $R^i : S \times A \times S \to \mathbb{R}$
- γ : discount factor, $\gamma \in [0, 1)$

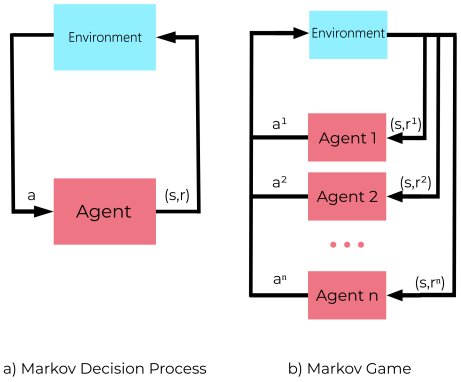


Fig. 1. Comparison of MDP and MG [21].

Figure 1 shows the differences between MDP and MG.
If all agents are homogeneous and exchangeable, it is called a Team Game or a Multi-Agent Markov Decision Process (MMDP). In this special case of MG, all reward functions are equal: $R = R^1 = ... = R^n$ [26].
Problems that can be described with a MDP or MG are fully observable. This means that the agents can observe all processes of their environment, i.e. the observation space is equal to the state space. In partially observable problems, the agents can only observe a certain area of its environment. Partially observable problems are the generalization of fully observable problems. The Partially Observable Markov Game (POMG) is the generalization of the MG and is defined with the tuple $(N, S, \{A^i\}_{i\in N}, P, \{R^i\}_{i\in N}, \{\Omega^i\}_{i\in N}, \{O^i\}_{i\in N}, \gamma)$ [27]. These variables extend the MG:

- Ω^i : set of observations of agent i, $\Omega := \Omega^i \times ... \times \Omega^n$ and $o \in \Omega$
- O^i : observation function, $O^i : S \times A \times \Omega \to [0, 1]$

A partially observable MMDP is defined with a Decentralized Partially Observable Markov Game (Dec-POMP) [26].
Another MG or POMG requirement is parallel execution of all actions. With an Agent Environment Cycle (AEC), actions can be executed sequentially. The AEC is defined with the tuple $(N, S, \{A^i\}_{i\in N}, \{T^i\}_{i\in N}, P, \{\mathcal{R}^i\}_{i\in N}, \{R^i\}_{i\in N}, \{\Omega^i\}_{i\in N}, \{O^i\}_{i\in N}, \gamma, v)$ [27]. For the AEC, the POMG definitions apply with the following adaptations:

- T^i : transition function of agent i, $T^i : S \times A_i \to S$
- P : transition function of environment, $P : S \times S \to [0, 1]$
- $\mathcal{R}^i$: set of rewards of agent i, $\mathcal{R}^i \subseteq \mathbb{R}$
- R^i : reward function of agent i, $R^i : S \times N \times A \times S \times \mathcal{R}^i \to [0, 1]$
- v : next-agent function, $v : S \times N \times A \times N \to [0, 1]$

According to [27], there is an AEC for each POMG and vice versa.
Figure 2 illustrates the dependencies of the described markov processes.

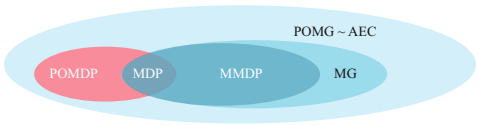


Fig. 2. Venn-diagram of markov processes [7], [26]: MDP ⊂ MMDP ⊂ MG ⊂ POMG and MDP ⊂ POMDP ⊂ POMG and POMG ∼ AEC.

C. MARL characteristics

MARL systems can be assigned to different categories based on their characteristics, such as agent interaction or MARL training architectures.
Based on the reward function, the interaction of agents can be divided into three groups [28]:

- Cooperative: A MARL is cooperative if the agents work together. If all agents share the same reward function $R^1 = R^2 = ... = R^N = R$ this is also called fully cooperative [23]. This particular function is mapped with the MMDP. The goal of fully cooperative behavior is to maximize the team reward [23].
- Competitive: Competitive behavior is said to occur when each agent tries to be better than the other agents. With the "zero-sum" MG one describes the special case of complete competition. It is described by $R = \sum_{i=1}^{n} R^i = 0$, i.e. the sum of the rewards of all agents is zero. Each agent minimizes the rewards of the other agents by maximizing its own [23].
- Mixed interaction: Mixed interaction is also known as "general-sum". It describes problems that are not uniquely cooperative or competitive. The relationship between the agents and the reward functions do not follow a fixed structure [21].

The exponentially growing set of states and actions with the number of agents poses a challenge for training MARL systems [23]. One consequence of this is the use of training architectures. According to [29], the training architectures describe often used structures. A distinction is made between training and execution. The training of MARL systems can again be divided into a distributed and a central approach [30]. On the other hand, execution takes place centrally or decentrally [23]. The training architectures are shown in Figure 3 and are divided as follows:

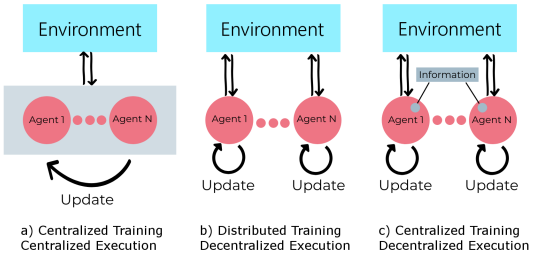


Fig. 3. Training architectures according to [23]

Decentralized training:

- Distributed Training Decentralized Execution (DTDE) describes an architecture in which each agent learns independently and autonomously. They do not exchange information, neither in training nor in execution [23].

Centralized training:

- Centralized Training Centralized Execution (CTCE) is a central architecture in which all agents share a common strategy. This strategy makes the decision for each individual agent both during training and execution. It is assumed that all agents can exchange information permanently and without limit [23].
- Centralized Training Decentralized Execution (CTDE) describes an architecture in which each agent has its own strategy. During training, the agents can exchange information that is no longer available at the time of execution. This concept can again be divided according to the type of agents. There are heterogeneous and homogeneous agents. The latter are characterized by similar structures, for example a same optimization goal. On the contrary, heterogeneous agents differ in their structure [23].
  "Parameter sharing" is called an exemplary method for homogeneous agents. It allows learning a strategy with the experience of all agents. Due to the same structures, the agents can use this strategy as their own during execution [31].

IV. Methodology

A connected factory is comparable to a network with many different components. These can be sensors, buttons or control units, for example. The network participants can send data to arbitrarily distant receivers in the network, on which these data are further processed. This comes along with the problems that have already been described in Section I. Intelligent edge computing depending on resource utilization, e.g. CPU or RAM, is intended to counteract this problem and is implemented below in the form of a MARL.

A. Solution approach

The MAS consists of independent subareas, the so-called agent zones. In these areas a local agent is responsible for one or more computing units. The agent decides whether and in which way data is processed. The MARL pursues the following goals:

- Resource utilization on computing units under specified threshold
- Uniform utilization of machines
- Maximization of the number of processed data
- Best possible processing of the data

This means that all agents must cooperate to achieve the goals. For collaboration, the agents communicate with each other and exchange necessary information. Thus, the decision of the agents remains local, despite the global objective. The architecture of the decentralized MARL is shown in Figure 4.

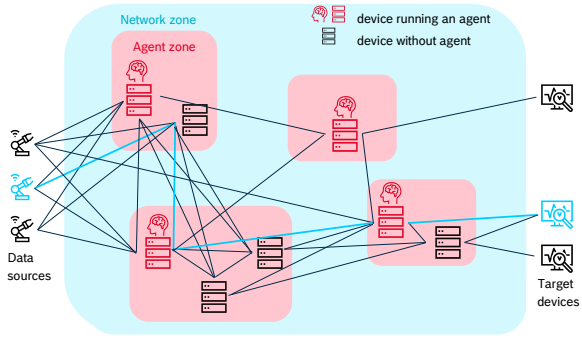


Fig. 4. Schematic presentation of the general solution approach [7]

The figure presents an exemplary IIoT network with multiple agent zones. The agent zone consist of an agent and at least one computing unit, which resources the agent allocates. The data processing tasks are executed on the edge devices within the agent zone according to the agents decisions. The data streams are forwarded and processed in the manner of a multi-hop approach. Thus, the amount of raw data is increasingly reduced due to the data processing and information extraction. The light blue line outlines an exemplary path of the data through different agent zones to the destination.

B. Boundary Conditions

The described vision of a MARL will be implemented under the following boundary conditions:

- Agent zones: Agent and computing unit are combined into one component. It follows that the agent observes its own resources and the data is processed on the same hardware.
- Permissions and availability: All data is available throughout the network and all network components have permission to read and process the data.
- Algorithms: The data can only be processed with one algorithm.
- Goals: The goal of each agent is to run its assigned resource below a specified threshold and balance the load across all agents. Not considered is the choice of the algorithm as in this study only one algorithm is available, and the maximization of the processed data is only indirectly considered.
- Resources: The agent considers only one resource (CPU) for optimization.
- Observability: For further problem description it is assumed that the environment is completely observable.

V. Concept

Analogous to ML classification, the original concept [6] was based on a multi-label approach. To reduce complexity, the first approach was adapted and led to following concept, which is comparable to a single-label approach.
Based on its own resource utilization and the utilization of the other agents, each agent decides locally to start an algorithm for data processing. The agent cannot stop the data processing again. All agents try to keep the resource utilization below the specified threshold g and to achieve a balanced utilization within the network in cooperation with the other agents. As all agents have the same tasks and goals, the problem can be described with a MMDP. Thus, they are homogeneous and interchangeable. The MMDP is described as follows [7]:

- Environment: The environment of the agents is an IoT network, because the data is to be processed on edge devices.
- Agent set: The number of agents varies depending on the size of the network.

$$N = \{1, 2, ..., n\} \text{ and } n > 1$$

- State set: The state describes the observed resource utilization and the average resource utilization of all agents. It holds:

$$s = \{s^1, s^2\} \text{ with } 0 < s^1 \leq 1 \text{ and } 0 < s^2 \leq 1 \text{ and } s \in S$$

$$\text{and } s^2 = \frac{1}{n} \sum_{m=0}^{n} s^{1,m} \tag{1}$$

- Action set of agents: There are two discrete actions, run algorithm or do not run algorithm. For action $a \in A$ applies:

$$A = \{0, 1\}$$

- Reward: All agents have the same reward function.

$$R(s, a) = R^1(s, a) = R^2(s, a)... = R^n(s, a) \tag{2}$$

The agent is penalized if it does not perform any actions or exceeds the threshold g. If it performs an action and is below the threshold g, it is rewarded.

$$R(s, a) = \begin{cases} \text{Penalty} & \text{if } a = 1 \text{ and } s_{t+1} > g \\ \text{Penalty} & \text{if } a = 0 \text{ and } s_{t+1} < g \\ \text{Reward} & \text{else} \end{cases} \tag{3}$$

VI. Experiments

For the experiments, the concept was implemented in a PettingZoo environment. PettingZoo[1] is a python library for MARL. The environment is a simulation of edge devices in an IoT network. A workload variable is defined for each edge device. The data flow is not simulated. For each data point a random workload value is defined, which can be added to the workload variables. The values are between 0 and 1 and represent in workload in percent. The MMDP was converted to an AEC that is used in the PettingZoo library. By the means of Stable-Baselines3 (SB3)[2], another python library, the environment was trained and the results are evaluated in the experiments. For experiments, the following values apply unless otherwise specified:

- SB3-MlpPolicy

[1] https://www.pettingzoo.ml/
[2] https://stable-baselines3.readthedocs.io/en/master/

- Training algorithm: Advantage Actor Critic (A2C)
- Total number of training steps: 1 000 000
- Training steps per episode: 100
- Learning rate: 0.0007 (A2C) or 0.003 (Proximal Policy Optimization (PPO))
- Number of agents: 2

1st Experiment: In the first experiment, different reward strategies are evaluated as listed in Table I.

TABLE I
VALUES OF REWARD FUNCTIONS FOLLOWING EQUATION 3

	$R_1(s,a)$	$R_2(s,a)$	$R_3(s,a)$
$a = 1 \ \& \ s > 0,8$	-10	-10	-1
$a = 0 \ \& \ s < 0,8$	-1	-10	-1
else	1	1	1

In Figure 5 can be seen that all three reward strategies motivate the agents to execute algorithms, but none of the chosen functions cause the agents to keep the CPU load under the threshold. Moreover, the machines of the first agent are often overloaded. The reward function R_1 leads to the largest fluctuations, either the agents do not execute any algorithm following the strategy or they overload the machines. R_2 and R_3 lead to relatively similar results for the second agent, with R_2 achieving better results for the first agent. For the further experiments, the reward function R_2 is chosen.

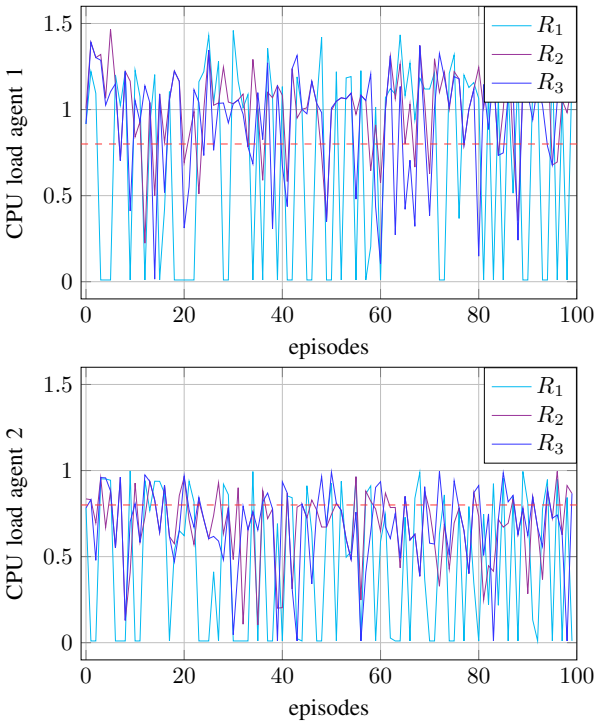


Fig. 5. Final CPU load per episode for different reward functions

2nd Experiment: The second experiment compares the best reward strategy of the first experiment to the policy optimized with PPO. For the RL algorithm PPO, the same parameter settings as for the A2C method are chosen. It is trained with the reward function R_2 of the first experiment.

For both learning algorithms selected, the training resulted in a policy that motivates both agents to execute algorithms (Figure 6).

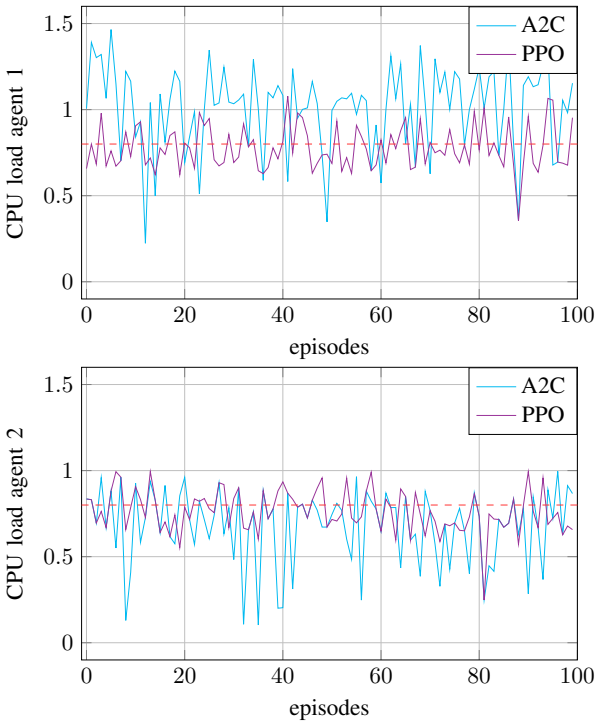


Fig. 6. CPU load per episode for different learning algorithms

The CPU load is either near or above the threshold for both agents at the end of the episodes. The A2C policy regularly leads the first agent into overload. Moreover, the policy leads to a larger fluctuation for both agents. The PPO method fluctuates less in the result and leads to a much better result for the first agent. The low peak-to-valley values of the PPO algorithm are due to its gradient method. For the further experiments PPO is used.

3rd Experiment: In the third experiment, the episode length was varied for training and evaluation. Both agents were trained with 10, 100 and 200 steps per episode.
A comparison of the result is shown in the two diagrams of Figure 7.

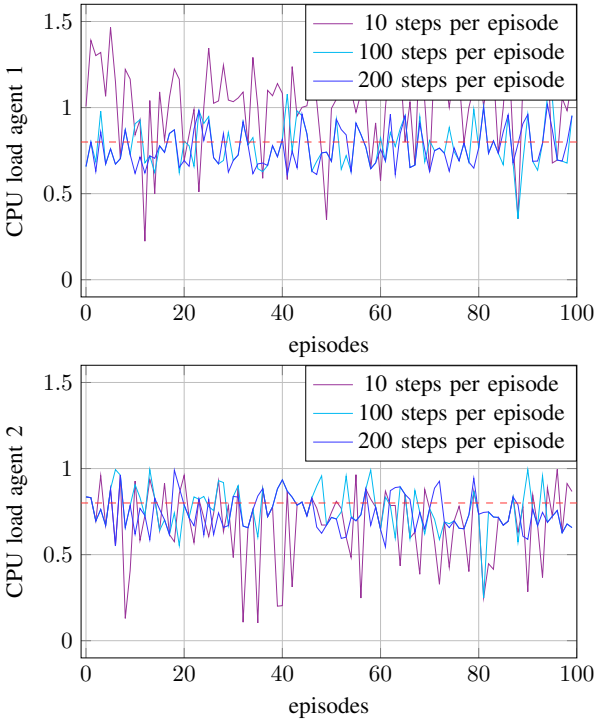


Fig. 7. CPU load per episode for different episode lengths

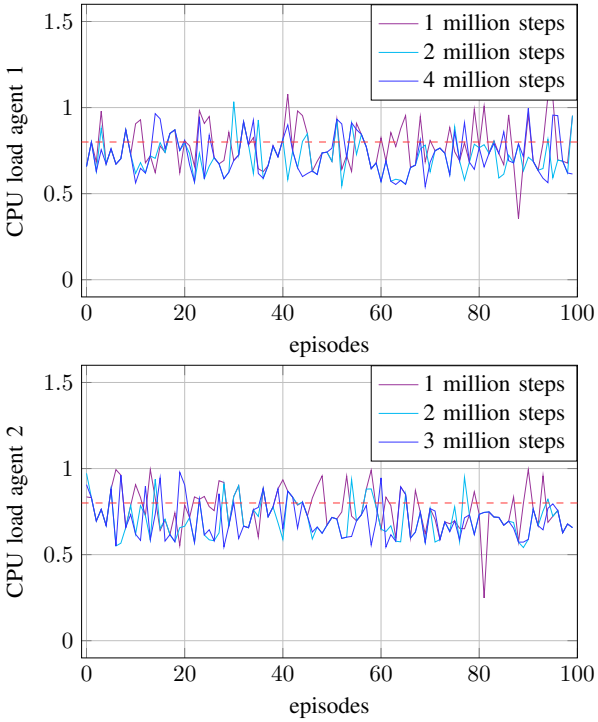
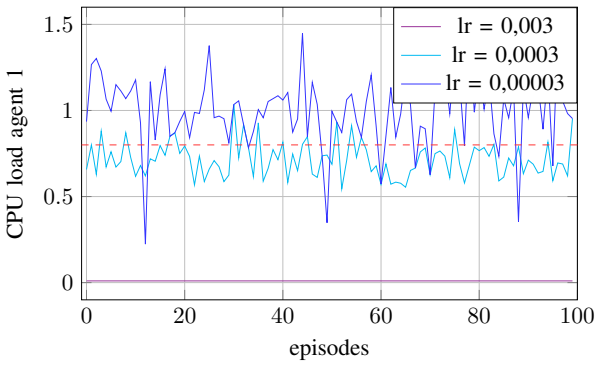


Fig. 8. CPU load per episode for different training lengths

All policies lead to the execution of algorithms and all violated the 80% threshold. Furthermore, the episode length of 10 steps leads to a larger variation in the result and the first agent overloads more often. From this can be concluded that the agents cannot learn the long-term effect of their actions well with this episode length. Increasing the episode length improved the results. As similar results are achieved for 100 and 200 steps per episode, the episode length of 100 is used for the further experiments.

4th Experiment: The fourth experiment investigates the training length. For this purpose, policies were trained with 1 million, 2 million, and 4 million training steps and subsequently evaluated over 100 episodes.
The different training lengths lead to similar results in the evaluation (Figure 8). All learned policies cause the agents to execute algorithms. For the most part, no agent overloads its computational unit. If about 30% of the final CPU loads are still above the threshold for the first agent, more than half are below the 80% mark for the second agent. Furthermore, it can be seen that 2 and 4 million training steps significantly reduce the fluctuations compared to 1 million. Moreover, 4 million training steps do not improve the policy much compared to 2 million. For this reason, 2 million training steps are used for the remaining experiments.

5th Experiment: Whereas previously the default SB3 PPO learning rate is used, this is changed in the fifth experiment. Three strategies, each with 2 million training steps and an episode length of 100 steps, are trained using the PPO method. The experiments differ in the learning rate between 0.003, 0.0003 and 0.00003. The evaluation in Figure 9 shows a clear result. While the two smaller learning rates lead to a policy that executes algorithms, with the learning rate 0.003 a policy was learned that does not allow any data processing. Looking at the evaluation of the first agent, it can also be seen that the learning rate of 0.00003 is not sufficient to find a similarly optimal policy for the same number of training steps compared to the learning rate 0.003.



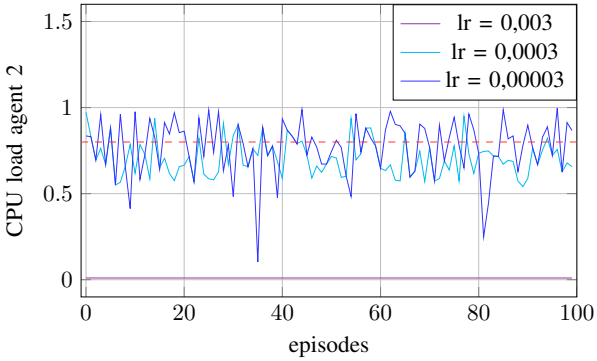


Fig. 9. CPU load per episode for different learning rates

6th Experiment: In the sixth experiment, the PettingZoo Environment (PEnv) was trained with five agents. Figure 10 compares the test run of two agents with the test of the five agents.

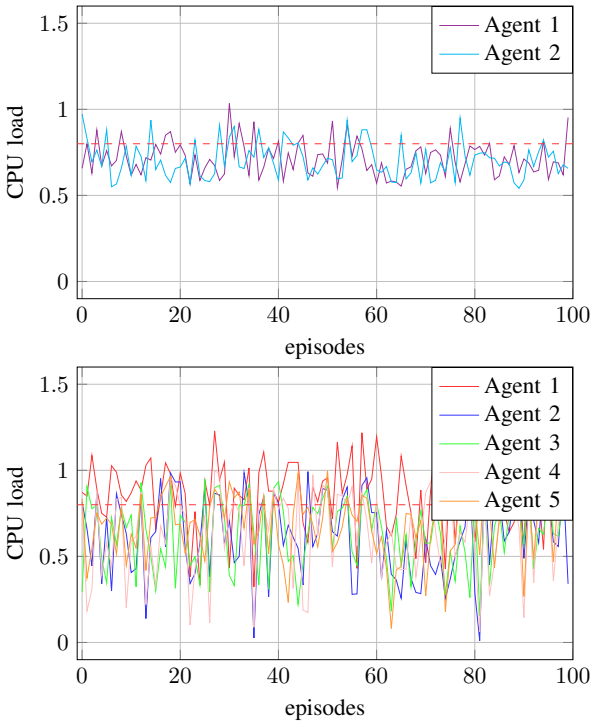


Fig. 10. Evaluation of scalability: CPU load per episode for number of agents

In both tests, the CPU loads fluctuate around the target value (threshold). The evaluation of the five agents also shows that all five agents behave similarly. This is primarily due to the CTDE training architecture.

VII. Discussion

The multi-agent implementation was evaluated and optimized with six experiments. Parameters were adjusted for training and a suitable reward function was found. Looking at the best result (Figure 11), it is noticeable that the CPU usage is consistently within the threshold range, but also exceeds the threshold value a couple of times and lead to overload one time. The test episode (lower diagram) shows that both agents quickly allocate the available computing resources and stop putting further load on the devices as soon as the threshold is reached.

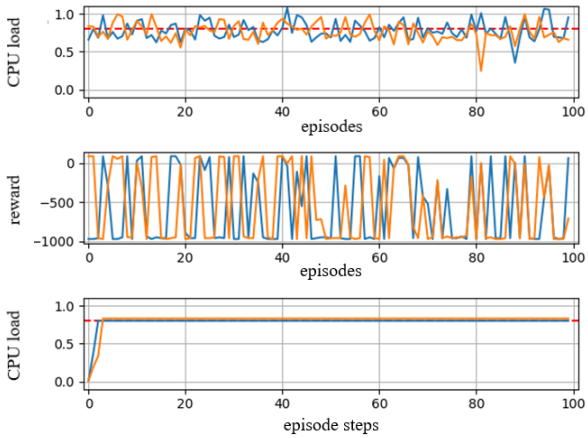


Fig. 11. Best result in experimental evaluation

The experimental evaluation highlighted the influence of the chosen hyperparameter values on the overall performance of the RL system. Finding the best hyperparameter values is a research field itself. For optimal results, the usage of hyperparameter optimization methods, e.g. basic methods like grid or random search or more complex methods like particle swarm optimization, is recommended.

The evaluation is limited to the training process of the MAS in a simulated environment. The simulated test environment is limited to static CPU base loads of the devices and does not consider any other processes with varying CPU load that are running on the edge devices. Furthermore, in this setup only one data processing algorithm is considered in resource allocation. Thus, the varying complexity of the processing tasks is not known by the agents yet. Further evaluations in a real industrial setup are required to make a statement about the performance in solving the task of resource allocation as well as the overhead due to the agent system itself and its applicability on industrial edge devices.

The algorithm itself is limited as it does not consider RAM or hardware memory usage. The agents are only able to decide about the execution of a task but not about ending a running task. Thus, the system is only able to increase the CPU load, but not to reduce it again.

VIII. Conclusion

In summary, this study describes detailed evaluation results on a new approach that uses MARL for resource allocation on edge devices in the IIoT. The experiments examine in particular various hyperparameter settings and show the complexity and relevance of good parameter choices. In the experimental evaluation in the simulated environment, the proposed method appears to be a promising approach for overcoming resource limitations and enabling edge computing. Based on the presented approach and results, two interacting MAS for the management of computing and communication resources in the IIoT has been elaborated, which is presented in detail in [7]. The article confirms the suitability and potential of the RL based resource allocation for Industry 4.0.

References

[1] I. Markit, "Internet of Things: a movement, not a market." https://cdn.ihs.com/www/pdf/IoT_ebook.pdf, 2017. Accessed: 2021-12-06.

[2] W. Shi and S. Dustdar, "The Promise of Edge Computing," *Computer*, vol. 49, pp. 78–81, 2016.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, (New York, NY, USA), p. 13–16, Association for Computing Machinery, 2012.

[4] M. Weiner, M. Jorgovanovic, A. Sahai, and B. Nikolić, "Design of a low-latency, high-reliability wireless communication system for control applications," *2014 IEEE International Conference on Communications (ICC)*, pp. 3829–3835, 2014.

[5] M. De Donno, K. Tange, and N. Dragoni, "Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog," *IEEE Access*, vol. 7, pp. 150936–150948, 2019.

[6] J. Rosenberger, M. Urlaub, and D. Schramm, "Multi-agent reinforcement learning for intelligent resource allocation in iiot networks," in *2021 IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT)*, 2021.

[7] J. Rosenberger, M. Urlaub, F. Rauterberg, T. Lutz, A. Selig, M. Bühren, and D. Schramm, "Deep Reinforcement Learning Multi-Agent System for Resource Allocation in Industrial Internet of Things," *Sensors*, vol. 22, no. 11, 2022.

[8] B. Vogel-Heuser, ed., *Softwareagenten in der Industrie 4.0*. De Gruyter Oldenbourg, 2018.

[9] *At Your Command!: Agentenbasiert Zur Smart Factory*. Essen: Vulkan Verlag GmbH, 2020.

[10] K. Manchella, A. K. Umrawal, and V. Aggarwal, "FlexPool: A Distributed Model-Free Deep Reinforcement Learning Algorithm for Joint Passengers and Goods Transportation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 2035–2047, 2021.

[11] L. Wang, X. Hu, Y. Wang, S. Xu, S. Ma, K. Yang, Z. Liu, and W. Wang, "Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning," *Computer Networks*, vol. 190, p. 107969, 2021.

[12] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of Deep Reinforcement Learning in Communications and Networking: A Survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[13] Y. Chen, Z. Liu, Y. Zhang, Y. Wu, X. Chen, and L. Zhao, "Deep Reinforcement Learning-Based Dynamic Resource Management for Mobile Edge Computing in Industrial Internet of Things," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4925–4934, 2021.

[14] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart Resource Allocation for Mobile Edge Computing: A Deep Reinforcement Learning Approach," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1529–1541, 2021.

[15] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource allocation based on deep reinforcement learning in iot edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1133–1146, 2020.

[16] X. Liu, J. Yu, Z. Feng, and Y. Gao, "Multi-agent reinforcement learning for resource allocation in iot networks with edge computing," *China Communications*, vol. 17, no. 9, pp. 220–236, 2020.

[17] Y. Ren, Y. Sun, and M. Peng, "Deep reinforcement learning based computation offloading in fog enabled industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4978–4987, 2021.

[18] Z. Cao, P. Zhou, R. Li, S. Huang, and D. O. Wu, "Multiagent deep reinforcement learning for joint multichannel access and task offloading of mobile-edge computing in industry 4.0," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6201–6213, 2020.

[19] P. Sun, Z. Guo, G. Wang, J. Lan, and Y. Hu, "MARVEL: Enabling controller load balancing in software-defined networks with multi-agent reinforcement learning," *Computer Networks*, vol. 177, p. 107230, 2020.

[20] T. Li, Z. Xu, J. Tang, and Y. Wang, "Model-Free Control for Distributed Stream Data Processing Using Deep Reinforcement Learning," *Proc. VLDB Endow.*, vol. 11, p. 705–718, feb 2018.

[21] K. Zhang, Z. Yang, and T. Basar, "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms," *CoRR*, vol. abs/1911.10635, 2019.

[22] G. Weiss, *Multiagent systems: a modern approach to distributed artificial intelligence*. Cambridge: MIT Press, 1999.

[23] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: a survey," *Artificial Intelligence Review*, 2021.

[24] L. S. Shapley, "Stochastic Games," *Proceedings of the National Academy of Sciences*, vol. 39, no. 10, pp. 1095–1100, 1953.

[25] M. L. Littman, "Markov Games as a Framework for Multi-Agent Reinforcement Learning," in *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ICML'94, (San Francisco, CA, USA), p. 157–163, Morgan Kaufmann Publishers Inc., 1994.

[26] Y. Yang and J. Wang, "An Overview of Multi-Agent Reinforcement Learning from Game Theoretical Perspective," *ArXiv*, vol. abs/2011.00583, 2020.

[27] J. K. Terry, B. Black, A. Hari, L. Santos, C. Dieffendahl, N. L. Williams, Y. Lokesh, C. Horsch, and P. Ravi, "PettingZoo: Gym for Multi-Agent Reinforcement Learning," *ArXiv*, vol. abs/2009.14471, 2020.

[28] L. Busoniu, R. Babuska, and B. De Schutter, "A Comprehensive Survey of Multiagent Reinforcement Learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.

[29] G. Bono, J. S. Dibangoye, L. Matignon, F. Pereyron, and O. Simonin, "Cooperative Multi-agent Policy Gradient," in *Machine Learning and Knowledge Discovery in Databases* (M. Berlingerio, F. Bonchi, T. Gärtner, N. Hurley, and G. Ifrim, eds.), (Cham), pp. 459–476, Springer International Publishing, 2019.

[30] G. Weiss, "Distributed Reinforcement Learning," in *The Biology and Technology of Intelligent Autonomous Agents* (L. Steels, ed.), pp. 415–428, Berlin, Heidelberg: Springer Berlin Heidelberg, 1995.

[31] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative Multi-agent Control Using Deep Reinforcement Learning," in *Autonomous Agents and Multiagent Systems* (G. Sukthankar and J. A. Rodriguez-Aguilar, eds.), (Cham), pp. 66–83, Springer International Publishing, 2017.

Authors

B. Sc. Michael Urlaub studies in the master program "Technische Informatik" at TH Köln.

Dipl. Ing. (FH) Julia Rosenberger is doing her doctorate at Bosch Rexroth AG in the field of "Datenflussoptimierung in der Industrie 4.0".

Investigations on self-optimizing PID controllers based on neural networks and Implementation in a process control system

André Wittling
Shell Energy and Chemicals Park Rheinland, Abt. DMR/R3M4 Process Automation
Ludwigshafener Straße 1, 50389 Wesseling, Germany
TH Köln, Fakultät F07, Institut für Nachrichtentechnik, Betzdorferstr. 2, 50679 Köln
a.wittling@shell.com

***Abstract*—The system developed in this paper aims at optimizing the control parameters of a PID controller. A neural network implemented within a process control system performs the optimization. The basic idea of the approach is based on the use of a simple neural network structure, which is characterized by a fast response time. This ensures a fast computation of the optimal control parameters. The training of the neural network is based on a numerical backpropagation algorithm using an advanced Levenberg-Marquardt algorithm. Additional knowledge about the plant and the control loop model is required to compute reasonable control parameters. The acquisition of this additional knowledge is illustrated using an example flow measurement. Furthermore, the aspect of loss of stability by adding a subsystem to the control structure based on bounded input bounded output stability is discussed.**

***Index Terms*—neural network, PID control, Levenberg-Marquardt algorithm, neural PID, closed-loop model, BIBO stability, system identification, Honeywell, vortex flow principle, plant-related asset management**

I. Introduction

Smart Factory is one of the goals being pursued within the German government's Industry 4.0 theme complex [1]. Smart Factory describes an intelligent process structure that manages itself without human intervention. Based on this idea, an increase in digitalization can be seen within the process industry. The use of intelligent systems and cheaper hardware components is leading to the increase in digitalization. The combination of more intelligent systems and cheaper hardware also leads to an increased amount of data. This data includes process data, which today is mostly unused [2]. Considering the high volume of data and with regard to advantageous information retrieval, e.g., to be able to make optimizations to processes, manual execution can be very time-consuming or too complex for a human. For this reason, a suitable alternative is to entrust the information retrieval to a machine. Particularly in recent years, it has become apparent that systems with implemented artificial intelligence are capable of processing large volumes of data very quickly and in an optimized manner.

For this reason, artificial neural networks are used to optimize given process control problems, e.g. [3] and [4]. The implementation of an artificial neural network used for control purposes comes with the cost of losing stability guarantees and interpretability of the control structure [5]. Furthermore, limitations of the process control system have to be considered, e.g., limited memory. Systems that serve to optimize control parameters must take these aspects into account. A check is therefore essential. In continuous processes, different operating points occur, resulting in different system dynamics. A setting of the associated control parameters is usually very extensive and is made for a defined operating point. As a result, optimum control behavior at varying operating points is not guaranteed.

II. Approach

The approach shown in this paper deals with the optimization of a given flow control loop. Based on related work, e.g., [5] - [7] an optimized control behavior can be achieved by extending the structure of the standard closed loop control. Therefore, a new subsystem is added, which is based on artificial intelligence (AI). On the one hand, the AI can be used for actual control of the plant [6], and on the other hand, it can be used for tuning purposes [5], [7].

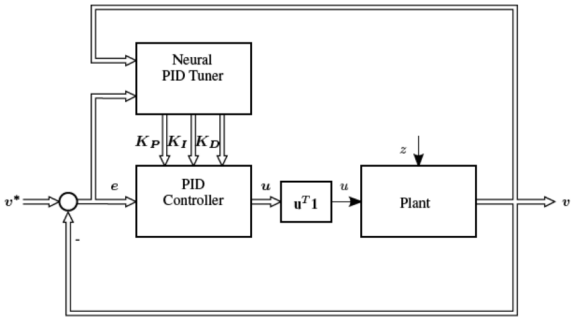


Fig. 1. Control structure with an additional neural network [5]

Fig. 1 shows the standard closed control loop structure consisting of a PID controller and the plant. Additionally, a neural

PID tuner can be identified. The neural PID tuner describes the additional AI-subsystem. In this case, the neural PID tuner handles the optimization of the control parameters during operation using a neural network. Following the approach of [5], the development of a neural PID tuner for the given flow control loop is shown in this paper. As a part of the development, each subsystem of the closed control loop requires its own considerations with respect to the functionality of the overall system.

III. PID CONTROLLER FUNCTIONALITY

A. Theoretical considerations of PID control

The most common implementation of control strategies is the PID control [8]. In recent implementations, a PID controller is based on an additive form of three terms. These terms are the proportional (P), the integral (I) and the derivative (D) term. Each term shows a different reaction to a control error e. The value of e is based on a difference between a reference variable v^* and the output of the plant v. The effect of e can be influenced by three control parameters. These are K_P, K_I and K_D as they are shown in Fig. 1. K_P is the proportional coefficient of the P term. This determines whether amplification or attenuation is to be applied based on the control error. The integration coefficient, K_I of the I term defines a delay time and K_D a reaction time of the system for the D term. Equations (1) to (3) express the general relationship between the control difference e and the three control parameters.

$$u_P(t) = K_P * e(t) \tag{1}$$

$$u_I(t) = K_I * \int e(t)dt \tag{2}$$

$$u_D(t) = K_D * \dot{e}(t) \tag{3}$$

The output of the P, I, and D terms are represented by u_P, u_I, and u_D. The final output u of the PID control is calculated by the sum of all three terms. Fig. 2 shows the detailed scheme of the PID controller.

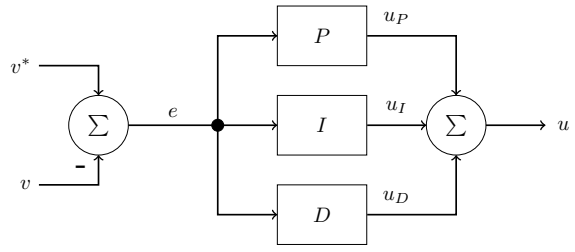


Fig. 2. A detailed view of an additive PID controller

Based on the detailed scheme u is calculated by (4).

$$u(t) = K_P * e(t) + K_I * \int e(t)dt + K_D * \dot{e}(t) \tag{4}$$

An alternative calculation is given by a multiplicative combination of the three terms. In older implementations, this combination of a PID control is used [9]. Therefore, a different scheme is defined for the functionality.

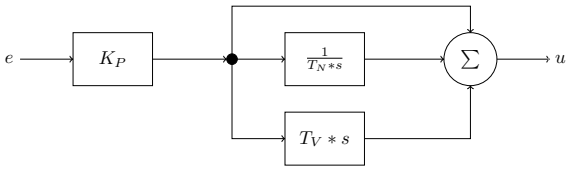


Fig. 3. Detailed view of a multiplicative PID controller

Fig. 3 shows the modified scheme of the PID controller. From the figure, the setting of the P term has an influence on the other two terms, resulting in amplification or attenuation of all effects on the control error e as a function of K_P. In addition, Fig. 3 shows two different control parameters for the I and the D term. Based on the quotient between the control parameters K_I, K_D and K_P the parameters T_N and T_V are defined.

$$T_N = \frac{K_P}{K_I} \tag{5}$$

$$T_V = \frac{K_D}{K_P} \tag{6}$$

Equations (5) and (6) show the relationship between the control parameters with T_N and T_V in seconds. T_N defines the reset time. The reset time is the time that a pure I controller would have to spend to achieve the same output value generated by a P controller at a constant control error. T_V is the derivative time and defines how much faster a pure D controller can generate the output of a P controller. Following the principle shown in Fig. 3 the output u is calculated using the following equation.

$$u(t) = K_P * [e(t) + \frac{1}{T_N} * \int e(t)dt + T_V * \dot{e}(t)] \tag{7}$$

B. PID Control within the Honeywell DCS

For the development of a self-optimizing PID controller, the functionality of the used Distributed Control System (DCS) needs to be known. Investigations regarding this paper are based on the DCS by Honeywell running on Release 501.6. The computed equation for the output of a PID controller is given by [10].

$$CV(t) = K * \mathcal{L}^{-1}[(1 + \frac{1}{T_1 s} + \frac{T_2 s}{1 + \alpha T_2 s}) * (PV_s - SP_s)] \tag{8}$$

Equation (8) shows the calculation of the control variable CV, which represents a step response for the actual output of the PID controller. The calculation is shown in the Laplace domain based on (7), where K represents K_P, T_1 represents T_N and T_2 represents T_V. Two further differences arise from consideration of (8). The first difference is the use of controller deviation $\hat{e}$. $\hat{e}$ is formed from the difference between the process value (PV) and the reference variable, which is also known as the setpoint (SP). It describes the negative control error e. The second difference can be seen in the calculation of the D term. The D term is formed by the Laplace

transformation of the controller deviation in combination with a low-pass filter. The low-pass filter uses a constant frequency, which is equal to a value of $\alpha = \frac{1}{16} = 0.0625$. By applying the Laplace inverse, the calculation of the step response results in (9).

$$CV(t) = K * (\frac{e(t)}{T_1} + 17e(t)\delta(t) - \frac{256e(t)e^{\frac{-16t}{T_2}}}{T_2}) \quad (9)$$

The effect of the Dirac pulse δ cancels out at any time where $t \neq 0$. Two further considerations lead to a final equation for the calculation of the step response. First, the two control parameters, T_1 and T_2, must be converted into seconds. Within the Honeywell DCS, these are specified in minutes. The second consideration is the relevance of the processing cycle. Equation (8) is valid only in cases where the processing cycle is equal to one second. If this differs, the result must be adjusted accordingly. This is done by multiplication with the process cycle in seconds.

$$CV(t) = (K * (\frac{e(t)}{60 * T_1} - \frac{256e(t) * e^{\frac{-16t}{60*T_2}}}{60 * T_2})) * t_{cycle} \quad (10)$$

The final equation for the step response is shown by (10) where t_{cycle} represents the mentioned process cycle time. The resulting output, $u(t + t_{cycle})$, is calculated by adding $CV(t)$ to $u(t)$. This equation is one of the requirements for the development of a training algorithm for a neural network that will optimize the PID controller.

IV. CONTROLLED SYSTEM ANALYSIS

The analysis of the properties of the controlled system leads to the topic of system identification. System identification offers different approaches, which represent the properties of the controlled system by a model. The model is only a representation of the system's behavior within defined quality criteria [11]. Data must be collected through interactions with the system. A suitable approach must be chosen based on the data and the background knowledge about the controlled system. Investigations for the implementation of a self-optimizing PID controller require two essential pieces of information about the controlled system. The first piece of information is based on the formation of the process variable. This affects the control difference and thus the controller output variable. Thus, a relationship between the process variable and the controller output variable is required. The second piece of information is used to prove stability. As mentioned in the introduction, the modification of the control loop does not lead to a guarantee of a stable system [5]. To prove the stability of the closed control loop, the transfer behavior between the controlled system and the PID controller needs to be known.

A. Model based on first principles

If the measurement method of the controlled system is known, it is possible to determine a model by using mathematical approaches [11]. The controlled system investigated in this paper is a flow measurement based on the vortex flow principle. A vortex flow measurement is based on the principle of Kármán vortex street. This principle states that a medium behind a disturbance body is formed by counter-rotating vortices.

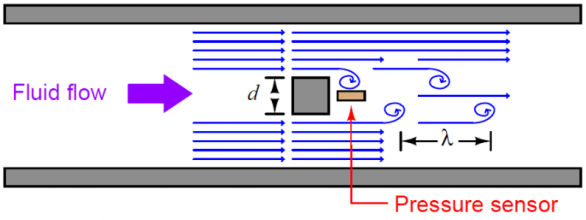


Fig. 4. Vortex flow measurement [12]

Fig. 4 depicts an example of vortex flow measurement. A medium inside a pipeline flows towards a disturbance body with dimensions of d. Behind the disturbance body, turbulences with a wavelength of λ occur, which form differential pressures due to their counter-rotation. The differential pressures lead to pressure pulses, which can be detected by a sensor. The sensor accumulates the pressure pulses and forms a vortex frequency f from them. The wavelength λ is related to the dimension of the disturbance body. The relationship is described by the shape of the oscillating flow. The swirl's shape is defined by the Strouhal number S_r. [12].

$$\lambda = \frac{d}{S_r} \quad (11)$$

The described relationship is illustrated by (11). According to the approaches from [13], the shape of the vortices is related to the inertial and viscous forces. These are described by the Reynolds number R_e. The Reynolds number includes the material properties of the medium, such as its viscosity and density. Following the principle of traveling waves using the frequency-velocity-wavelength formula, the medium velocity is defined by (12).

$$v = \frac{f * d}{S_r(R_e)} \quad (12)$$

The controlled system under investigation defines the mass flow of the medium as the process variable. The mass flow is determined by the flow velocity within a body. This body is described by a pipeline whose cross-section D corresponds to the surface of a circle. In addition, when forming a mass flow, the temperature- and pressure-dependent density ρ must be taken into account. The result of these correlations is shown in (13).

$$\dot{m} = \rho(T, p) * \frac{\pi * D^2 * f * d}{4 * S_r(R_e)} \quad (13)$$

By using this equation and performing a series of measurements, a relationship between the controller output variable and the process variable can be established. The correlation is based on the determination of the vortex frequency. The series of measurements for the investigated control system is given in table I.

TABLE I
MEASUREMENT RESULTS FOR THE CONTROL SYSTEM

y in %	$\dot{m}$ in t/d	v in m/s	R_e [10^5]	S_r	f in Hz
0	0	0	0	0	0
15	350.46	0.59	0.45	0.197	1.61
20	458.39	0.77	0.59	0.197	2.11
35	830.83	1.41	1.08	0.197	3.83
50	1298.95	2.20	1.68	0.197	5.99
60	1502.82	2.55	1.95	0.197	6.93
84	2098.172	3.56	2.72	0.197	9.68
100	2267.4	3.85	2.94	0.197	10.46

The table shows the mass flow as a function of the regulating variable y. In this case, the manipulated variable corresponds to the controller output variable u. The regulating variable represents a valve position and shows the percentage value of the opening. The associated mass flow has been detected. The measurement series covers a period of 500 days. The value shown is the mean value of this period. By solving (13) for the frequency and inserting it into (12), the flow velocity can be determined. With the help of the flow velocity, the Reynolds number and, from this, the Strouhal number can be determined using the approach from [14]. The value of the Strouhal number in most applications is approximately 0.2, which is also shown by the results. The vortex frequency is to be calculated from the determined values.

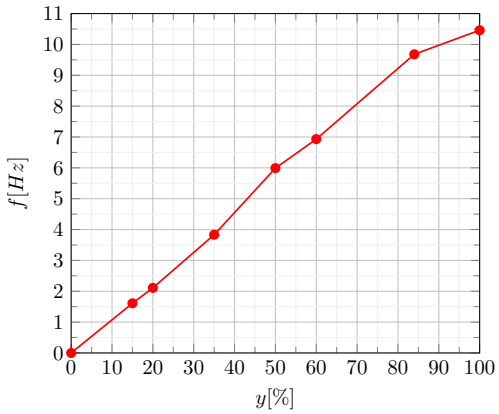


Fig. 5. Correlation between the manipulated variable and the vortex frequency

Fig. 5 shows the relationship between the regulating variable and the determined frequency. The curve is almost linear. For this reason, linear interpolation and the establishment of a straight-line equation are acceptable for this controlled system. The vortex frequency f can thus be defined as a function of y.

B. Determine the transfer behavior

Various approaches exist for determining the transfer behavior, such as the inflectional tangent principle, which e.g. leads to a PT1 substitute model [9]. The cycle in which the data is provided by the DCS plays an important role. Based on this cycle, the choice of a suitable procedure must be made. Under the consideration of the Honeywell DCS, this results in the provision of data in one-second increments.

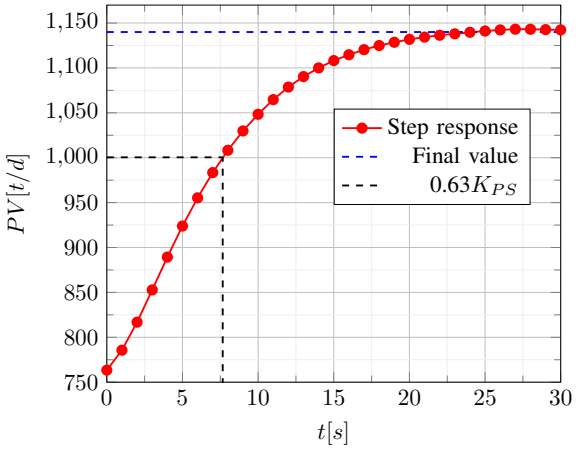


Fig. 6. Exemplary step response of the process variable

Fig. 6 shows an exemplary step response of the controlled system. The shown PV corresponds to the process output v. Based on the progress of the PV, the system behavior of a PT1-term can be identified. By means of this observation, a PT1 substitute model can be used for the representation of the plant. The calculation rule for the PT1-term in the time domain is given in (14) [15], where T is the fixed delay time and K_{PS} is the proportional factor.

$$v(t) = K_{PS} * (1 - e^{\frac{-t}{T}}) \tag{14}$$

Given that K_{PS} is the step response's final value, the fixed delay time T must be determined. Because of the functionality of the PT1 term shown in (14), T can be determined at the time when $t = T$. Using this relationship, v at this moment corresponds to the value of $0.63K_{PS}$. For the example shown in Fig. 6, this results in a fixed delay time of T of approximately 7.66 seconds. The investigations of the controlled system by various step responses lead to comparable results. In these results, T is equivalent to a time span of 7.98 seconds on average.

Using the PT1 substitute model, the transmission behavior of the closed control loop can be determined. Therefore, the calculation rule of the PT1-term in the Laplace-domain is needed, which is shown by (15).

$$V(s) = \frac{K_{PS}}{1 + Ts} \tag{15}$$

The transmission behavior of the closed control loop is generally described by

$$G(s) = \frac{G_e(s) * G_r(s)}{1 + G_e(s) * G_r(s)} \tag{16}$$

where G is the transmission behavior based on G_e and G_r. The derived substitute model G_e is given by (15). G_r

represents the behavior of the used controller. For the system under consideration, this is given by the behavior of the PID controller in (8). Substituting the two equations into (16) yields a final equation for the transmission behavior

$$G(s) = \frac{s^2 * b_2 + s * b_1 + b_0}{s^3 * a_3 + s^2 * a_2 + s * a_1 + a_0} \tag{17}$$

where each coefficient is given by table II.

TABLE II
COEFFICIENTS OF THE TRANSMISSION FUNCTION

Coefficient	Value
b_2	$K_{PS}K * (\alpha T_1 T_2 + T_1 T_2)$
b_1	$K_{PS}K * (T_1 + \alpha T_2)$
b_0	$K_{PS}K$
a_3	$\alpha T_1 T_2 T$
a_2	$\alpha T_1 T_2 + T_1 T + \alpha K_{PS} K T_1 T_2 + K_{PS} K T_1 T_2$
a_1	$T_1 + K_{PS} K T_1 + \alpha K_{PS} K T_2$
a_0	$K_{PS}K$

V. THE ARTIFICIAL NEURAL NETWORK

Due to the previous considerations, all the necessary information about the process is already known. This information can be used to choose an approach for an artificial neural system. The choice of the approach depends on the possibility of performing online or offline training of the system. By performing online training, the artificial neural system acts directly with the PID controller. Possible approaches of this type are, e.g., a single neuron PID controller [6] or the combination of a fuzzy system with an artificial neural network [7]. On the one hand, these use the discovered properties of the PID controller to take over the actual control or to optimize the control parameters. For the investigated flow measurement, the application of online training is not possible due to the effects on the overall process. For this reason, only the possibility of an artificial neural system for a self-optimizing PID controller based on an offline training approach is given.

A. Definition of a network structure

As an approach for the implementation, the idea of the neural PID tuner [5] from Fig. 1 has been followed. The neural PID tuner, proposed in [5], consists of a General Dynamic Neural Network (GDNN), which belongs to the type of feedback networks.. The recurrent connections are delayed by one time step and are randomly generated. They are used for learning non-linearity and representing system dynamics. In its basic structure, the GDNN describes a simple multi-layer perceptron (MLP) with one hidden layer. Simple in this context means that the number of neurons in the hidden layer is very small. The neural network's inputs are the control difference e and the process variable v, as shown in Fig. 1. The outputs correspond to the control parameters. This results in an input layer of two neurons and an output layer of three neurons. A property of recurrent connections is that they need some time steps to reach a steady state. This does not guarantee the required stability of the neural network [9].

A modified approach has been chosen to avoid this behavior. This approach does not consider the system dynamics of the recurrent connections. This means that the artificial neural network of the PID Tuner is described by a simple, fully connected MLP.

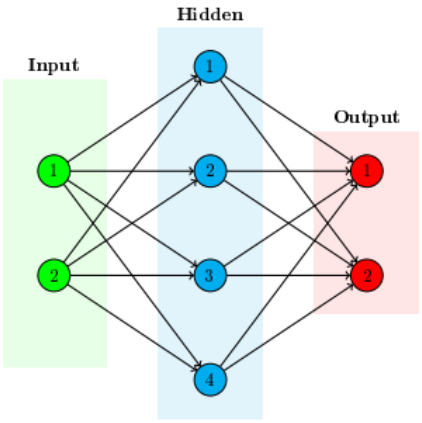


Fig. 7. Exemplary artifical neural network for the PID Tuner

Fig. 7 shows a possible MLP that can be used to implement a neural PID tuner. Each neuron corresponds to the structure of an MPC neuron [16]. The hidden layer consists of four neurons. This small number allows a fast computation of the network outputs. These are shown with two neurons. The omission of the third neuron is due to Schlitt's adjustment rules [17]. These adjustment rules state that the use of a PI controller is sufficient for controlling a flow. Furthermore, typical setting ranges for the control parameters are defined.

The activation functions of the neurons are different for each layer. According to the approach of the GDNN, the activation function of the neurons in the input layer is the identity function [5]. The activation function of the hidden layer is chosen as tangent hyperbolic [18]. The operation of the PID controller is crucial for the choice of the activation function of the output layer. Based on the considerations for the Honeywell DCS, a multiplicative combination of the control parameters occurs. This means that the control parameters of the I and D terms are defined as times. These can't be negative numbers. For this reason, the absolute value function is used for the activation function of the output layer. The implementation within the DCS is carried out in two steps. The training and testing phases of the system are to be performed separately from the DCS. The background of this separation is the higher computational load and needed storage, which are necessary for the execution of the subsequent epochs. The goal of the first step is to obtain a neural network that provides reasonable control parameters. The second step is the implementation within the DCS.

B. Execution of the offline training

The basis of offline training is the Levenberg-Marquardt (LM) algorithm [19]. This method describes a training method based on backpropagation. The mode of operation is based on the gradient descent method and the Gauss-Newton method and describes the updating of the weights of the neural network by

$$w_{k+1} = w_k - (J_k^T * J_k + \lambda * I)^{-1} * J_k * E_k \tag{18}$$

where w defines the weights of the neural network, J the Jacobian matrix, I the identity matrix, E the error function and k a discrete moment in time. λ defines a damping factor, which determines to which method the LM algorithm converges. If $\lambda \ll J_k^T * J_k$, the LM algorithm will converge to the Gauss-Newton method. Otherwise, if$\lambda \gg J_k^T * J_k$, the LM algorithm will converge to the gradient descent method. As it can be seen from (18), the Jacobian matrix must be determined to calculate the new weights. The elements of this matrix contain the first partial derivative of the network error after each weight of the neural network, considering each training pattern. This results in time-consuming computations, depending on the number of training patterns and network outputs [20]. To minimize the computation time, an approximated Jacobian matrix $\hat{J}$ can be obtained using finite difference [21]. Each element of $\hat{J}$ is calculated by

$$j_i = \frac{v(x,w) - v(x, w - h\varepsilon(w_i))}{\varepsilon(w_i)} \tag{19}$$

where j_i is the i-th Jacobian element of $\hat{J}$, v is the process variable as a function of weights and a given input x, and ε is the step size for the i-th weight w_i. h describes a column vector containing the values zero and one. This allows a controlled weight modification of w using $\varepsilon(wi)$. To create a finite difference, the step size must be as small as possible. For this purpose, the significand precision value for double precision numbers can be used, which is approximately $1.1*10^{-16}$ [22]. From (19) emerges another essential aspect for the training of the neural network. As mentioned before, the general Jacobian matrix in the context of the LM algorithm includes the partial derivative of the network errors according to the weights. The network error is composed of the difference between the expected value and the network output when using the backpropagation algorithm. In the case of the PID controller, this means that the optimal control parameters must be known. Since these are not known, the backpropagation algorithm must be modified. This leads to a numerical consideration in which training is not based on the network error but on the control difference. The control difference is defined as the difference between the PID controller's given setpoint and a calculated process variable v as a function of the weights w and network inputs combined into the input vector x. The calculation of $v(x, w)$ can be done by using (10) and (13). Based on the control parameters, which correspond to the outputs of the neural PID tuner, a weighted y must be calculated. Based on Fig. 5, the vortex frequency f can be defined as a function of y. By means of this relationship, the resulting frequency in (13) yields $v(x, w)$.

The last required consideration is based on providing the system dynamic. A system dynamic is essential for training of the time-relevant control parameters. For this reason, the training is performed over a growing sequence of training patterns, which is generated during the training process. Static processing of the training patterns leads to the fact that the system tries to compensate the control error only with a pure P-controller. The computation of the offline training can be organized as pseudocode, as shown in Fig. 8.

```
// Start of the training algorithm
init neural network
// Iteration over training data
for each entry in training data
add entry to training sequence seq
    // Training of the neural network over ep epochs
    for ep epochs
        calculate pidgains using neural network
        calculate v(x,w) using (13)
        for each entry in seq
            for i weights
                calculate j_i using (19)
            end for i weights
        modify weights using (18)
        if ∑(v(x,w) − SP)² < ∑(PV − SP)²
          transfer weights to neural network
          modify λ by λ = λ/2
        else
          modify λ by λ = λ * 2
        end for each entry in seq
    end for ep epochs
end for each entry in training data
// End of the training algorithm
```

Fig. 8. Pseudocode of numerical backprogation for a neural PID Tuner

C. Execution within the Honeywell DCS

The realization of the trained neural network within the DCS is done in a Custom Algorithm Block (CAB). A CAB describes a user-defined function block, which in this case consists of three steps. The first step is the initialization of the neural network. For this purpose, the weights of the offline training that have been identified as optimal have to be stored. The weights also determine the structure of the network. The second step is the calculation of the network outputs, which leads to the new optimal control parameters by transferring the controller deviation and the process value. The third step takes up the observations of the transmission behavior of the closed loop in regards to the stability of the overall system. For the proof of bounded input bounded output (BIBO) stability, the Routh criterion is used for this application [23]. The routh criterion can be verified using the coefficient of the characteristic polynomial of the transfer function. Based on

(17), an associated Routh scheme is obtained, which is shown in table III.

TABLE III
ROUH SCHEME FOR THE PROOF OF BIBO STABILITY

a_0	a_2	0
a_1	a_3	0
$\frac{a_2 * a_1 - a_0 * a_3}{a_1}$	0	0
a_3	0	0

By using the coefficients from table II, the computation of the Routh scheme depends on the control parameters. After a performed computation of the new control parameters, the elements of the first column of the Routh scheme must be assessed. The system is considered stable if all its elements are not negative. The control parameters are accepted only if stability is given [23].

VI. RESULTS OF THE OFFLINE-TRAINING AND TESTING

During offline training, different structures of the neural network have been investigated. The different structures are based on a variation in the number of neurons in the hidden and output layers. With respect to the output layer, this results in training sessions for a PID and a PI controller. The variation of the neurons in the hidden layer served to determine a network structure that could reproducibly generate reasonable setting ranges of the control parameters for the given controlled system. For the system under consideration, five neurons are to be used in the hidden layer. The results of the training depending on the training patterns and the epochs for an example neural network are shown in table IV.

TABLE IV
RESULTS OF THE TRAINING PHASE OF A NEURAL NETWORK

patterns	epochs	K	T_1	T_2
10	20	0.03 ... 1.53	0.02 ... 0.7	0.02 ... 0.17
100	10	0.07 ... 0.93	0.18 ... 0.71	0.09 ... 0.13
100	20	0.12 ... 0.51	0.11 ... 0.62	0.06 ... 0.11
1000	10	0.13 ... 0.89	0.09 ... 0.59	0.06 ... 0.10
3743	5	0.11 ... 0.72	0.14 ... 0.51	0.004 ... 0.08

Two different criteria must be evaluated to assess the results in table IV. The first criterion is that the results are within or close to the setting range given by Schlitt [17]. For a flow control, a typical value for K is between 0.5 and 1.0 and the value for T_1 is between 0.1 and 0.5 minutes. From [17], it follows that the setting of a D-term is not useful. The second criterion is a similar setting to the existing PID controller. This has very good performance at the operating point. The corresponding settings are $K = 0.22$ and $T_1 = 0.2$ minutes.

If this is considered, the displayed setting ranges already assume meaningful values for a small number of training patterns and epochs. The setting ranges shown in table IV represent the determined minimum and maximum value for each individual control parameter based on the training patterns. It follows from the results that by using a larger number of training patterns, the network learns for the training of a PID controller that the D term is not useful. This results in a convergence towards zero for T_2. Based on these observations, it is sufficient to implement a neural PI tuner for the given controlled system. The reasonable adjustment ranges are due to the iteration of the epochs. Due to the steadily increasing sequence of training patterns and the subsequent execution of the epochs, a high total number of iterations by which the neural network can adapt to the system dynamics is shown. The total number of iterations is given by

$$I_n = \sum_{n=1}^{N} n * ep \tag{20}$$

where I_n represents the number of iterations, n represents the number of training patterns, and ep represents the number of epochs. For the last entry in table IV, this results in a training consisting of 35034480 iterations.

The test phase is used to identify possible configurations to be considered for implementation within the DCS. For this purpose, the extreme values of the networks are to be determined, which define the possible setting range of the control parameters. If possible, the test patterns should include the complete measuring range of the associated control system. In a log file, all network configurations are stored with the related weight factors and possible adjustment ranges, as shown in Fig. 9.

```
Weights:
[0.2642528 -0.24353593 0.01985194 0.20896227 0.19065475 -0.1238825
0.0885282 -0.20980556 -0.19545562 -0.25471893 -0.00380722 0.16504383
0.09048049 -0.04402695 0.2191904 0.01452629 0.42961403 -0.16358957
-0.26062988 -0.17386749]
Kmin= 0.2927440032704005 Kmax= 0.37676436117322215
T1min= 0.11385999 T1max= 0.2048214501514961
```

Fig. 9. Sample of the Log-File with the configuration of the neural PI controller

As a result of the test phase, various configurations have been identified as useful.

VII. RESULTS OF THE OPERATION PHASE

The goal of the operation phase is to compare the performance of the fixed-parameter PI controller with that of the neural PID tuner under similar conditions. For this reason, the neural PI tuner needs to be evaluated in terms of its performance during normal closed-loop operation. The operation phase of each individual AI-subsystem covers five consecutive days.

A. Evaluation criteria

During the five-day operation phase, data in one-second time steps has been logged. Each data entry consists of the

date and time as well as the PV and the SP. To evaluate the performance of the fixed PI controller and the neural PI tuner, two criteria were checked during the investigations. The first criteria is the mean squared error (MSE) of the day. The MSE is calculated by

$$MSE = \frac{1}{M} * \sum_{m=1}^{M} (v_m - v_m^*)^2 \tag{21}$$

where M is equal to the sample size. For one day, M equals 86400 samples. A low value for the MSE indicates good performance of the controller.

The second criterion is based on a valuation method linked to the theme complex of plant-related asset management (AM). As part of the plant-related AM, process values or technical documents are assessed to ensure value-preserving and value-enhancing maintenance [24]. One component of the AM is the verification of a controller's performance using control performance monitoring (CPM) methods. CPM methods are used by various DCS-system vendors, e.g., Siemens or Emerson [25]. In this method, the control performance of the closed loop is evaluated using various dimensional numbers. These dimensional numbers can be calculated using the current process value over time. The first dimensional number describes the variance c^2 of the PV, which is given by

$$c^2 = \frac{1}{Q-1} * \sum_{q=1}^{Q} (v_q - \bar{v})^2 \tag{22}$$

$$\bar{v} = \frac{1}{Q} * \sum_{q=1}^{Q} v_q \tag{23}$$

where v equals the PV, $\bar{v}$ is the average of v calculated by (23) and Q is the evaluation period. Since in normal operation, a dynamic for the SP can be identified, the evaluation horizont Q equals 300. This represents a time span of five minutes. For the determination of good control quality, the variance needs to be compared to a control loop, which is represented by a small variance. Those control loops are described as minimum variance controller (MV-controller) [26]. In theory, a MV-controller is based on a high-order system model, which results in high requirements for a mathematical model of the plant. For this reason, the MV-controller has not received much relevance in industrial practice [9]. Instead, an approximation of a control loop with MV-controller can be made by the process value of the real system [27]. The approximated control loop's variance c_{MV}^2 is defined as

$$c_{MV}^2 \approx c_{diff}^2 * (2 - \frac{c_{diff}^2}{c^2}) \tag{24}$$

where c_{diff}^2 describes the variance between two adjacent time steps of v. The calculation includes all samples from the evaluation horizon and is defined by (25).

$$c_{diff}^2 = \frac{1}{2 * (Q-1)} * \sum_{q=2}^{Q} (v_q - v_{q-1})^2 \tag{25}$$

On the basis of the dimensional numbers, the control performance can be evaluated by the computation of the control performance index (CPI). By using the results of (22) and (24), the CPI is computed as follows:

$$CPI = (1 - \frac{c_{MV} + \Delta c}{c + \Delta c}) * 100\% \tag{26}$$

In this calculation rule, Δc represents numerical stability in relation to the measuring range. The value is fixed and equals the value of 22 t/d for the following evaluation. This value represents 0.1% of the measuring range. A low CPI indicates good control quality with regard to MV-control.

B. Evaluation Results

During the evaluation, three different configurations of the neural PI tuner were investigated. As mentioned above, the evaluation period includes five consecutive days. The same evaluation period of the fixed PI controller has been taken into account. One of the best five-minute segments of the fixed PI controller is shown in Fig. 10.

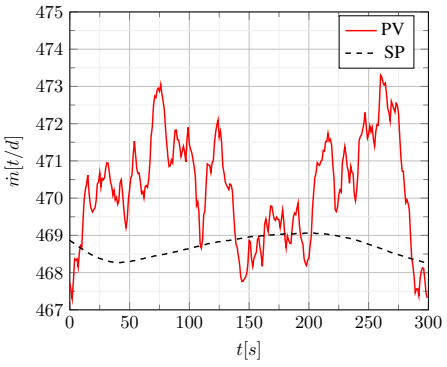


Fig. 10. Time trace of the fixed PI controller

As it can be seen in Fig. 10, the PV mostly overshoots the desired SP with an average of 2 t/d. Besides that, a small segment in the time interval of 130..200 seconds can be noticed, where the controller has an acceptable control performance with a control error of 1 t/d as a maximum. The MSE of the whole segment results in 4.62 t^2/d^2. The MSE for this segment is low overall, as is the CPI, which is 4.6915 %. By definition, the control quality of this segment is acceptable for proper control functionality.

In comparison, the average settings of the control parameters as well as the MSE and the CPI of each individual PID tuner can be seen in table V.

TABLE V
SETTING RANGES AND BEST PERFORMANCES OF THE PID TUNER

PID tuner	K	T_1 in min	MSE in t^2/d^2	CPI in %
1	0.26924	0.77156	1.28	3.9321
2	0.26129	0.58968	0.93	3.5084
3	0.25174	0.30845	0.64	1.8559

The shown values represent the best five-minute segments during the investigation. The associated time traces can be seen in Fig. 11 to 13.

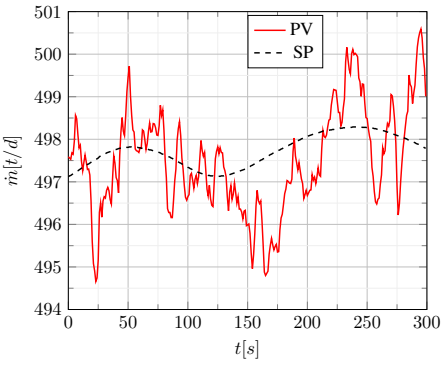


Fig. 11. Time trace of the first neural PI tuner

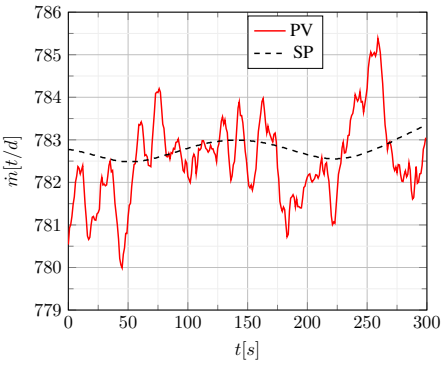


Fig. 12. Time trace of the second neural PI tuner

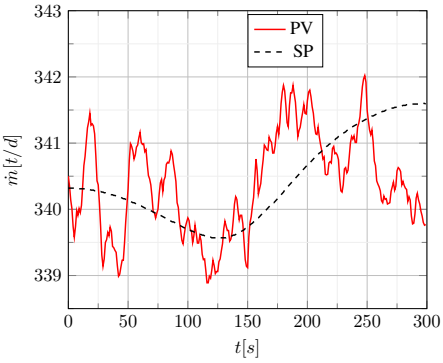


Fig. 13. Time trace of the third neural PI tuner

Within the time traces, an improvement in the control quality can be noticed by a decreasing control error in comparison to the time trace shown in Fig. 10. This is proven by the values of the MSE and CPI given by table V. But to prove an improvement in the control quality by extending the closed control loop with a neural network, the overall performance of the five-day operation phase needs to be compared. The results are shown in table VI.

TABLE VI
RESULTS OF THE OPERATION PHASE

Day	fixed		Tuner 1		Tuner 2		Tuner 3	
	CPI	MSE	CPI	MSE	CPI	MSE	CPI	MSE
1	11.3	18.3	10.7	11.8	8.2	11.5	11.8	14.3
2	11.1	30.9	10.5	16.8	10.5	13.7	10.1	9.9
3	18.3	36.3	9.1	11.0	11.4	16.9	10.2	10.5
4	17.5	30.9	10.5	13.0	10.6	11.7	9.0	9.3
5	14.0	20.3	10.5	15.6	9.8	11.2	10.6	11.3
Av.	14.4	25.2	10.3	13.6	10.1	13.0	10.0	11.1

As it can be seen within the results, each neural PI tuner provides a comparable CPI to the fixed PI controller. The increase in the control quality is shown by the five-day average in the last row of table VI. For all neural systems, the improvement is approximately 3%. On the other hand, a big decrease of the MSE can be noticed. This means that the small improvement of the CPI leads to an overall better control performance. In comparison, the average CPI of each neural PI tuner is approximately equal. Based on the MSE, the third PI tuner can be established as the control system with the best control performance. In addition, this result proves the assumption that an improvement in the overall control performance can be achieved by extending the closed control loop with a trained neural network based on archived data.

VIII. CONCLUSION

The approach described in this paper shows an attempt to identify optimal controller settings based on unused process data. The presented implementation shows that a neural network can learn a system dynamic from only a few data sets. The resulting optimal controller settings describe reasonable ranges that correspond to a typical setting. Due to the operating phase within the actual DCS, those reasonable setting ranges have been proven to be a valid solution for the increase of the control performance. In addition, the execution of the reduced algorithm of the CAB proves to be neither time-intensive nor does it necessitate a large amount of data storage. This means the shown approach is suitable for implementation within a DCS system. Ensuring system stability is possible by deriving from the closed loop transmission behavior. By calculating the Routh scheme, a safe adaptation between the PID controller and the neural network is given.

With a view to further work, this approach is to be tested for more complex systems, which e.g., include non-linearities. Here, a decisive factor is whether the necessary information from the controlled system can be derived.
Due to the easy extensibility of the algorithm, it is possible to optimize systems consisting of different PID controllers, e.g.,

controller cascades. Another approach is to conduct further training phases. The newly acquired data from the operation phase is to be used as the data basis. Thus, a new generation of artificial intelligence can be trained based on the best control performance.

IX. ACKNOWLEDGMENTS

First, I would like to thank Prof. Dr. Bartz for the continuous support, for sharing his expertise, and for his intellectual guidance during this work. I am grateful for the opportunity he has provided me.
Second, I would like to thank Mr. Meyer for sharing his knowledge and the helpful discussions.
I also want to express my gratitude to Mr. Friße from Shell Energy and Chemicals Park Rheinland for providing the investigated control system and for making the data acquisition and operating phase possible.

REFERENCES

[1] bmbf.de, Industrie 4.0, [Online], accessed 05-2022, Available: https://www.bmbf.de/bmbf/de/forschung/digitale-wirtschaft-und-gesellschaft/industrie-4-0/industrie-4-0.
[2] welt.de, Künstliche Intelligenz: Industrie sieht Deutschland bereits abgehängt, [Online], accessed 11-2021, Available: https://www.welt.de/wirtschaft/article191212599/Kuenstliche-Intelligenz-Industrie-sieht-Deutschland-bereits-abgehaengt.html.
[3] IK. Pirabakaran and V. M. Becerra, PID autotuning using neural networks and model reference adaptive control: International federation of automatic control, 2002.
[4] J. de Jesús Rubio, Discrete time control based in neural networks for pendulums, Applied Soft Computing Journal, vol. 68, pp. 821-832, 2018.
[5] J. Günther, E. Reichensdörfer, P. M. Pilarski, and K. Diepold, Interpretable pid parameter tuning for control engineering using general dynamic neural networks: An extensive comparison., PLoS ONE, vol. 15, no. 12, pp. 1-17, 2020-12-10.
[6] J. Liu, On a method of single neural pid feedback compensation control. online, 2016.
[7] Y. Yongquan, H. Ying, and Z. Tao, To tune the dynamic parameters of neural pid controller by the intelligent learning algorithm, in IEEE International Conference Mechatronics and Automation, 2005, vol. 3, pp. 1521-1526 Vol. 3, 2005.
[8] Ang KH, Chong G, Li Y. PID control system analysis, design, and technology. IEEE Transactions on Control Systems Technology. 2005;13(4):559–576.
[9] N. Große, W. Schorn, R. Bartz, N. Becker, and M. Kluge, Taschenbuch der praktischen Regelungstechnik: mit 44 Tabellen. München [u.a.]: Fachbuchverl. Leipzig im Carl-Hanser Verl., 2006.
[10] H. I. Sàrl, Control Builder Components Theory. Honeywell Honeywell International Sàrl, Feb. 2018.
[11] L. Ljung, System identifcation: theory for the user. Prentice Hall information and system sciences series, Upper Saddle River, NJ: Prentice Hall, 2. ed. ed., 1999.
[12] instrumentationtools.com, What is a Vortex Flow Meter? [Online], accessed 05-2022. Available: https://instrumentationtools.com/what-is-a-vortex-flowmeter/
[13] E. Achenbach and E. Heinecke, On vortex shedding from smooth and rough cylinders in the range of Reynolds numbers 6x103 to 5x106, Journal of Fluid Mechanics, vol. 109, p. 239-251, 1981.
[14] B. Sundan, Vortex Shedding. [Online], accessed 05-2022, Feb. 2011. Available: https://thermopedia.com/de/content/1247/
[15] G. Schwarze, Bestimmung der regelunsgtechnischen Kennwerte von P-Gliedern aus der Übergangsfunktion ohne Wendetangentenkonstruktion, messen-steuern-regeln Heft 5, S. 447-449, 1962, 1962.
[16] W. S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biology, vol. 52, no. 1-2, pp. 99-115, 1990.
[17] V. Schlitt, Herbert, Regelungstechnik in Verfahrenstechnik und Chemie, 1978.
[18] B. Kalman and S. Kwasny, Why tanh: choosing a sigmoidal function. online, 1992.
[19] D. W. Marquardt, An algorithm for least-squares estimation of nonlinear parameters, Journal of the Society for Industrial and Applied Mathematics, vol. 11, no. 2, pp. 431-441, 1963-06-01.
[20] B. Wilamowski and H. Yu, Improved computation for levenberg-marquardt training, IEEE Transactions on Neural Networks, vol. 21, no. 6, pp. 930-937, 2010-06-01.
[21] K. Zhou, J. Hou, H. Fu, B. Wei, and Y. Liu, Estimation of relative permeability curves using an improved levenberg-marquardt method with simultaneous perturbation jacobian approximation, Journal of Hydrology, vol. 544, pp. 604-612, 2017.
[22] IEEE, Iso/iec/ieee international standard - foating-point arithmetic. online, 2020-05-08.
[23] V. Unbehauen, Rolf, Systemtheorie. München [u.a.]: Oldenbourg [Neubearb. ab 7. Aufl.]., 1997.
[24] NE 129: Plant Asset Management. NAMUR, Leverkusen 2009
[25] Becker, N., Grimm, W. M., Piechottka, U.: Aspekte des Real Time (Process Equipment) Performance Monitoring (RTPM). Automatisierungstechnische Praxis (atp) 46(2004), S. 24-30.
[26] Unbehauen, H.: Regelungstechnik III. Vieweg Verlag, Braunschweig 1993.
[27] Blevins, T., McMillan, L., Wojsznis, K., Brown, W. K.,: Advanced Control Unleashed. ISA, Research Triangle Park 2003.

Visual detection of a charging station and implementation of a docking routine performed by an autonomous vehicle

David Kliewe
Computer Engineering - Communications Engineering
TH Köln - Cologne University of Applied Sciences
Cologne, Germany
david.kliewe@gmail.com

***Abstract*—This paper was written in the scope of a research project which was executed to investigate the visual detection of landmarks in camera-captured images. The OpenCV library, which provides functions for computer vision, was used to perform image processing on an already implemented Raspberry Pi of a vehicle. Therefore a PiCamera was added to the vehicle and connected to the Raspberry Pi. A charging station, represented by a green rectangle visual landmark, should be recognized in wide range via image-processing and steering actions as well as motor outputs should be adjusted to position the car towards the charging station. In the near-field docking process, an already developed infrared LED-Phototransistor system is reused to provide information about the angle of the car with regard to the station. The car should center its alignment to the charging station and perform the docking process autonomously. When the battery of the car is connected to the charging station successfully, a CAN-Message is sent to the integrated CAN-Bus to inform all other components.**

I. Introduction

According to a Deloiette study, autonomous driving will become established in 2035 [1]. But even today, it is already possible to have autonomous motor vehicles participate in public road traffic on certain defined routes. In addition, the amount of battery-powered vehicles increased over the last years and will increase in the future [2]. Indeed, all vehicles have to be manned when participating the public roads and the pilot also is responsible for the batteries to be charged. But there are different sectors where the usage of unmanned autonomous driving battery-powered vehicles increases (e.g. warehouse operation, deliveries).

The combination of unmanned autonomous vehicles which are batterie-powered leads to the problem, that the vehicles are responsible for their batteries to be charged timely to prevent themselves from becoming incapable of acting. Conclusively, in this research project a vehicle is to be extended, which is to be able to detect a charging station and to adjust its steering and driving actions to connect its battery to the charging station. For this purpose, different signals and information are used for the wide- and close-range docking process. In the following report, the implemented code, which includes the usage of OpenCV for image processing and charging station detection, is shown partly. Furthermore the necessary hardware extensions that were made as well as the results achieved during the project are presented.

II. Existing autonomous vehicle and its components

The existing vehicle as shown in Figure 1 is the main object of investigation in this project. The modular construction of the vehicle allows students to realize different projects and theses. The vehicle consists of four different Printed Circuit Boards (PCBs), which all have their own spectrum of tasks. One PCB acts as a power-management and provides various voltage levels for ICs, actors and sensors.

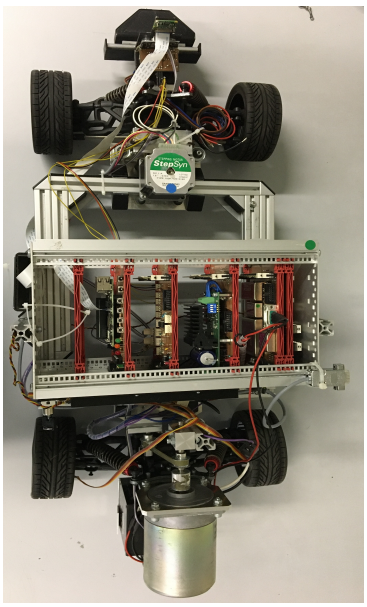


Fig. 1: Topview of vehicle and its modular structure

One acts as a sensor-PCB and processes all output-signals of different sensors attached to it. Corresponding to the sensor-PCB there is one actor-PCB, which manages the communication with motor drivers for steering- and drive-motors.
The fourth PCB is the so called RosPi-PCB. This PCB got its name from a set of software libraries called Robot Operating System (ROS) and its built in Raspberry Pi. This PCB enables the wireless communication to the vehicle. All PCBs can be modified and replaced according to different projects.
The autonomous vehicle is equipped with a CAN-Bus and all PCBs of the vehicle participate in the CAN-communication. They share information of sensor-signals and steering actions on the CAN-Bus and react to messages with certain identifiers. CAN-messages can be sent from external participants, either by external wire-interface, or by being sent to the Raspberry Pi via wireless LAN using the WebROS application. The Raspberry Pi acts as a gateway and forwards the received information to the CAN-Bus.

III. Comparison of Raspberry Pi and Jetson Nano

Since the visual detection of a charging station is the main part of this research project, it is relevant to implement a processing unit which is capable of image capturing and image processing. The most common single-board computers which can be used for this are the Raspberry Pi 4 as well as the NVIDIA Jetson Nano. TABLE I shows the main features of both single-board computers.

Feature	Raspberry Pi	Jetson Nano
CPU	Quad core 64-bit ARM-Cortex A72 1.5GHz	Quad-Core 64-bit ARM-Cortex A57 1.43GHz
GPU	/	128-core GPU 921 MHz
RAM	max. 4GB DDR4	max. 4 GB DDR4
Power-consumption	3 - 5 W	5 - 10 W
Networking	Ethernet WLAN Bluetooth 5.0, BLE	Ethernet

TABLE I: Comparison of Raspberry Pi and Jetson Nano [6] [7]

Both include UART-, I2C- and SPI-Interfaces, as well as GPIOs with PWM functionalities. They also include a 2-lane MIPI CSI-Interface for the attachment of a Raspberry Pi camera [6] [7]. Although the Jetson Nano has an onboard GPU which is useful for training and execution of ANNs and image processing, the hardware advantages including the low power-consumption of the Raspberry Pi in combination with the already built RosPi and implemented WebRos outweigh the features of the Jetson Nano. The RosPi-PCB is relevant for wireless communication which makes testing and error handling more easy than with an attached cable. By using the Raspberry Pi it is possible to attach a camera to the vehicle, to capture images of the environment and to process the images with OpenCV in an acceptable amount of time.

IV. Sensors and systems for docking process

A. Camera: Rasp Cam v2

The main sensor for the visual detection approach is the Raspberry Pi camera. The Raspberry Pi camera module includes a resolution of 8-megapixel. The maximum image transfer rate is 30 fps for 1920 x 1080 pixels, 60 fps for 1280 x 720 pixels and 60 fps for 640 × 480 pixels [8]. The most fluid image-stream is provided when using the resolution of 640 x 480 pixels. The horizontal field of view of the camera Module v2 is 62.2 degrees. The vertical field of view is 48.8 degrees [8].

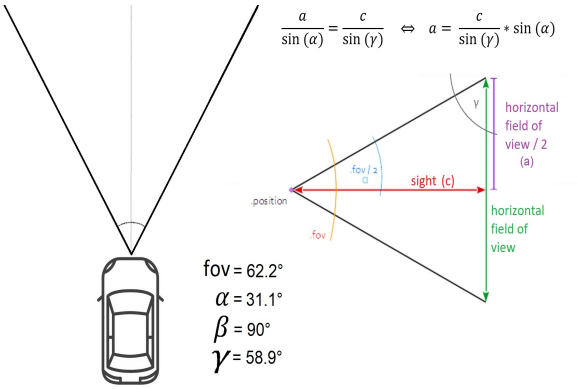


Fig. 2: Horizontal field of view of Raspberry Cam v2

Referring to Figure 2, the camera covers a horizontal area of 6 meters when capturing objects in a distance of 5 meters:

$$a = \frac{c}{sin(\gamma)} * sin(\alpha) = \frac{5m}{sin(58,9)} * sin(31,1)$$

Changing degrees-unit to radiant:

$$a = \frac{c}{sin(\gamma)} * sin(\alpha) = \frac{5m}{sin(1,028)} * sin(0,5427973974) = 3m$$

$$horizontalfieldofview = 2 * a = 2 * 3m = 6m$$

The camera is mounted and elevated at the front of the vehicle by a small 3D-printed holder for capturing images of the environment (see Figure 8). The camera is connected to the MIPI-CSI-Interface of the Raspberry Pi at the RosPi-PCB using a Raspberry Pi Camera flexcable.

The graphical user interface for the operating system ”Raspbian GNU/Linux 10 (buster)” running on the Raspberry Pi is installed via ssh. Therefore, the RosPi-PCBs powerconnection is established and the wireless network named ”ROSnet” is joined. The ssh connection to the Raspberry Pi is established via ssh-terminal (see Figure 3).

The sequence of commands shown in Listing 1 is executed on the Raspberry Pi via ssh.

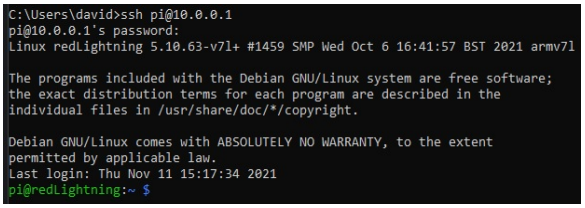


Fig. 3: Establishing ssh connection

```
sudo apt-get install update
sudo apt-get install upgrade
sudo apt-get install lightdm
sudo apt-get install raspberrypi-ui-mods
sudo apt-get install xrdp
```

Listing 1: Installation of GUI for Linux-OS

In addition to the installed desktop version, display-manager and user interface, the graphical login needs to be enabled by executing

sudo raspi-config.

Using the Remote Desktop Connection on a windows-computer, a RDP-connection to the Raspberry Pi can be established by connecting to the IP-address of the Pi (10.0.0.1) after joining the ROSnet wireless network. The installation of the GUI simplifies the implementation of the Raspberry Pi camera and is useful for upcoming debugging purpose. The next step for image capturing is the installation of python 3.7.3 and OpenCV 4.5.3 on the Pi. OpenCV provides a real-time optimized Computer Vision library and many useful tools, that can be used for image-capturing, image-processing and image-output operations.

```
import cv2
import time

# Initialize frame rate calculation
frame_rate_calc = 1
freq = cv2.getTickFrequency()

# Open a camera for video capturing
cap = cv2.VideoCapture(0)

def getImg(displayImage=False, displayFps=False):
        global t2
        global frame_rate_calc
        # Start timer (for calculating frame rate)
        t1 = cv2.getTickCount()
        success, image = cap.read()
        if displayFps:
                # Draw framerate to corner of frame
                cv2.putText(image,'FPS:␣{0:.2f}'.format(
                    ↪ frame_rate_calc),(30,50),cv2.
                    ↪ FONT_HERSHEY_SIMPLEX,1,(0,0,0)
                    ↪ ,1,cv2.LINE_AA)
        if displayImage:
                cv2.imshow("Image", image)
                cv2.waitKey(1)
                # Calculate framerate
                t2 = cv2.getTickCount()
                time1 = (t2-t1)/freq
                frame_rate_calc= 1/time1
        return image

while True:
        image=getImg(True, True)
```

Listing 2: Capture-Image function

The defined getImg() function (see Listing 2) consists of 2 input parameters. One parameter controls wether the captured image is displayed in a window or not. The second parameter controls wether the framerate should be calculated and displayed on the captured image or not. The function also returns the captured image for further processing. When setting both parameters to True and calling the function continously, captured images are displayed with about 30 fps.

B. Ultrasonic Distance Sensor

The HC-SR04 ultrasonic module is suitable for distance measurement in the range between 2cm and approximately 3m with a resolution of 3mm [9]. It's use is obstacle detection in the area. The measurement and thus the detection of an object has to be configured accordingly.

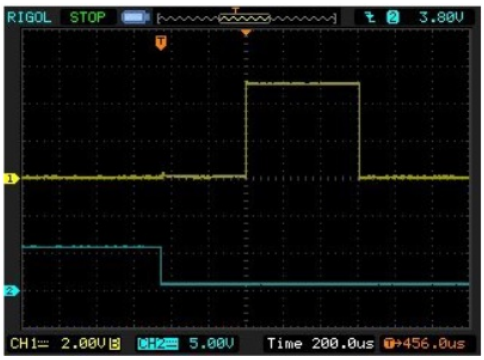


Fig. 4: Trigger Signal Ultrasonic Distance Sensor

Single Shot Mode: Figure 4 shows a single measurement. Channel 2 represents the trigger signal and channel 1 represents the output/input of the ultrasonic-signal. Setting the signal-level of channel 2 from HIGH to LOW initiates a measurement. After sending out echo-signals, the signal-level of channel 1 goes HIGH. After receiving the reflected ultrasonic-signals, the level on channel 1 goes LOW. The transit time of the ultrasonic burst in this measurement is about 600µs (200µs/div). The displayed time of 456µs is composed of from the 250µs delay after triggering and the subsequent 200µs burst.

Evaluation of the measurement: The speed of sound in air of 343 $\mathrm{m/s}$ (at 20°C) is now used to calculate the distance to the measurement object. 343 $\mathrm{m/s}$ corresponds to 34.3 cm per millisecond. The measurement in Figure 4 shows 600µs

(0.6 ms) between sent ultrasonic-signals and the received echo.
$\frac{34,3cm}{s} \cdot 0,6ms$ results in a signal-travel-distance of 20.6 cm. Since it is an echo, the distance is traversed twice, which is why the value needs to be divided by 2. For further information: [9].

There is one HC-SR04 ultrasonic module mounted at the front and at the back of the vehicle. For each sensor, a measurement is taken every 60 ms. This is the max. trigger frequency for the signals to not being influenced by a returning echo signal [10]. The ultrasonic-distance-measurement in the front and the back of the car is initiated by the sensor-PCB. The sensor-PCB calculates the distances and provides the information on the CAN-Bus. The SensDistanceUS-CAN-message with the Identifier 0x38 includes data of the distance to obstacles in the back of the car and in the front of the car in cm (see Figure 5).

Identifier	Sender	DLC	Data0 MSB	Data1 MSB	Data2 MSB	
0x038=56	Sens	3				
			x			Reserved
			x			Reserved
			x			Reserved
			x			Reserved
			x			Reserved
			x			Reserved
			x			Reserved
			x			Reserved
				M x x x x x x L		Distance front; in cm; 0xFF: >=255cm
					M x x x x x x L	Distance back; in cm; 0xFF: >=255cm

Fig. 5: SensDistanceUS-CAN-message

The messages provided on the CAN-Bus can be viewed using PCAN-View (see Figure 6).

CAN-ID	Type	Length	Data	Cycle Time
038h		3	000 087 056	60.0

Fig. 6: PCAN-View showing distances of front and back US-Sensors in cm

C. Actuators

The vehicle is able to perform movements due to its drive-motor which is controlled by a MD03 - 24Volt 20Amp H Bridge Motor Drive. The microcontroller TMS320F28335 on the actor-PCB uses I^2C-communication to write and read the register of the motor driver [11]. The 12 V DOGA DC Motor DO16941132B09/3060 attached to the motor driver then rotates and moves the vehicle corresponding to the output of the motor driver.

Steering actions are controlled by AMIS-30543 Micro-Stepping Motor Driver. The Motor Driver outputs signals to the SANYO DENKI steppermotor 103H7123-0140 which is attached to the steering axis with a drivebelt. The motordriver for steering actions is controlled by the microcontroller TMS320F28335 on the actor-PCB via SPI.

D. Charging station

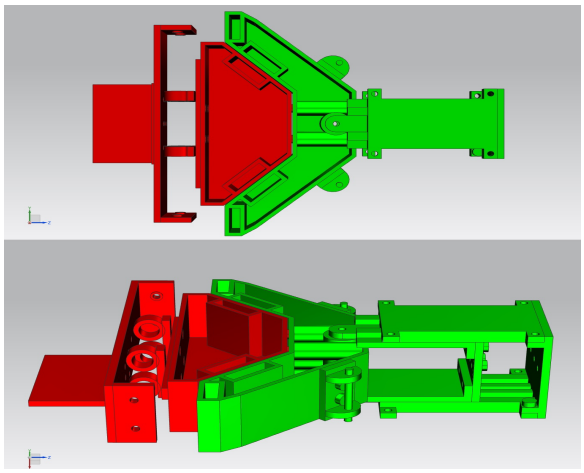


Fig. 7: Docking Station [5]

Figure 7 shows the construction of the already developed parts of a docking station. The red part is connected to the front of the vehicle, whereas the green part is connected to the charging system and has a fixed location. To be able to create a charging-connection, it needs to be ensured, that both parts of the docking station have the same road clearance. In order to make docking smoother, both parts include suspensions in different directions [5].
For the wide range detection and localization of the charging station, a green rectangle displayed on a tablet-pc is mounted on top of the charging station. For the short range detection and localization, infrared-LEDs are added to the charging station (see Figure 9). By amplifying and interpreting signals of photodiodes mounted at the front of the vehicle, it is possible to detect the distance and position of the vehicle towards the charging station (see Figure 8) [5].

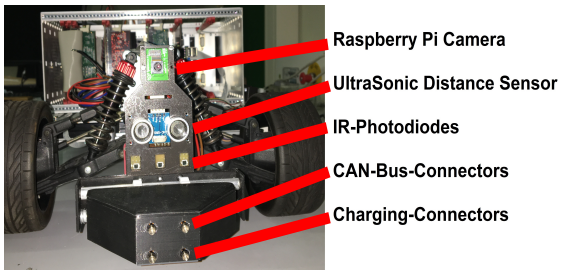


Fig. 8: Vehicle Front Sensors

The IR-LEDs of the charging station are controlled by the microcontroller TMS320F28335. In a period of 20 ms, the LEDs emit pulses with following sequence (see Figure 10 a)):
1) All LEDs emit pulse for 5 ms
2) All LEDs turned off for 3 ms

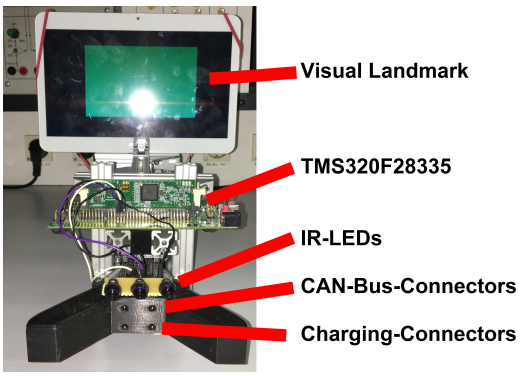


Fig. 9: Charging Station

3) Left LED (red) emits pulse for 1 ms
4) All LEDs turned off for 3 ms
5) Mid LED (blue) emits pulse for 1 ms
6) All LEDs turned off for 3 ms
7) Right LED (green) emits pulse for 1 ms
8) All LEDs turned off for 3 ms

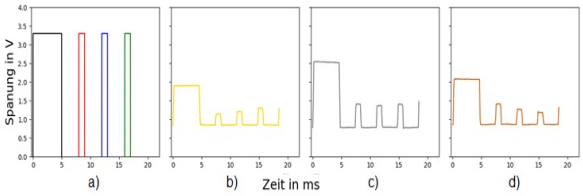


Fig. 10: IR-LED Output Sequence [5]

The mid-photodiode mounted on the front of the vehicle receives and amplifies the signals. As one can see in Figure 10 b), c), d), the values of the pulses vary depending on the orientation towards the charging station. The distance between vehicle and charging station is 30 cm for b) c) and d) [5]. The orientation in b) is 6 cm lateral to the right and the angle is 15° towards the charging-station, the orientation in c) is no lateral shift and the orientation in d) is 6 cm lateral to the left and the angle is 15° towards the charging-station (see Figure 11). The signals of the mid-photodiode are processed by the sensor-PCB and the information is then sent to the CAN-Bus for the near field docking process (see section VI-B).

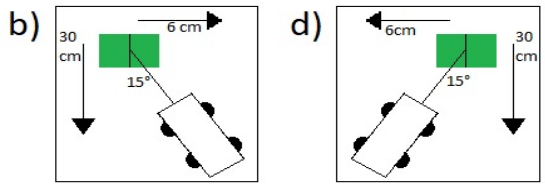


Fig. 11: Alignment of vehicle towards charging station

V. Visual Detection

A. Image processing using OpenCV

After capturing an image with the camera (see section IV-A), the image needs to be processed in a manner that the landmark (green rectangle) of the charging station can be recognized. Therefore the OpenCV library and its functions are used:

```
while True:
        image=getImg(False, False)
        #process image with opencv
        #convert image to hsv
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

        #set boundaries of hsv values
        lower_green = np.array([50, 70, 25])
        upper_green = np.array([80, 255, 255])
        #create mask with boundaries
        mask_green = cv2.inRange(hsv, lower_green, upper_green)
        #bitwise and captured image and green area
        result_green = cv2.bitwise_and(image, image, mask=
            ↪ mask_green)

        cv2.imshow("Image_thresh", result_green)
        cv2.imshow("Image", image)
        cv2.waitKey(1)
```

Listing 3: Detect green pixels in captured image

First, the BGR-image is converted into an HSV-image, where HSV is the abbreviation for Hue, Saturation, Value (or Brightness). For more information see [14].
A lower- and an upper-boundary with one value for Hue, Saturation and Value in each boundary is specified. Referring to Figure 12, the lower boundary with [50,70,25] and the upper boundary with [80,255,255] is selected to match the respective values for the green landmark.

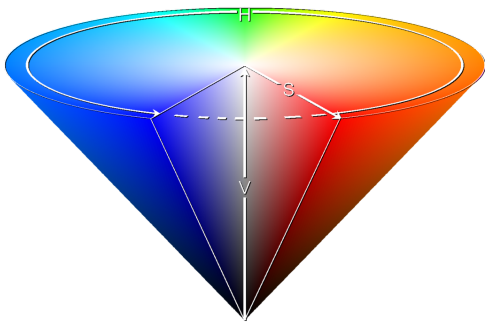


Fig. 12: Hue Saturation Value Cone [13]

The *cv2.inRange()*-function returns an image that shows pixels with value 255 (white), where the pixels of the HSV-image are inbetween the specified boundaries. All other pixels are set to 0 (black). For more information see [14].
The bitwise_and()-function of OpenCV connects the original image with the returned image of the *cv2.inRange()*-function.

The bitwise_and()-function ”calculates the per-element bit-wise logical conjunction for two arrays when src1 and src2 have the same size” [14].

Figure 13 shows the result of the implementation.

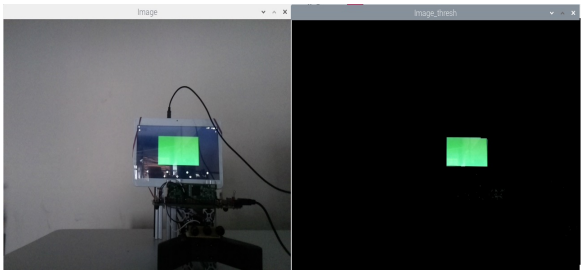


Fig. 13: Original and thresholded image

The darker the environment, the better the recognition, because the display of the tablet mounted on the charging station mirrors and reflects incident light.
The steps for thresholding are outsourced to the function *thresholdImg()*.
The next step after extracting the pixels that belong to the charging station is to determine where the charging station is located in the image. As this information is used to approach the charging station, it needs to be processed to drive left, right or straight.

```
while True:
        #Step 01: Capture Image
        image=getImg(False, False)

        #Step 02: Process image with OpenCV
        result_green = thresholdImg(image)

        #Step 03: Greyscale and blur image -->reduce noise
        gray = cv2.cvtColor(result_green, cv2.
            ↪ COLOR_BGR2GRAY)
        blurred = cv2.GaussianBlur(gray, (5, 5), 0)
        thresh = cv2.threshold(blurred, 130, 255, cv2.
            ↪ THRESH_TOZERO)[1]

        #Step 04: find contours in the thresholded image
        cnts, hierarchy = cv2.findContours(thresh.copy(), cv2.
            ↪ RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)

        for c in cnts:
                # compute the center of the contours
                M = cv2.moments(c)
                if M["m00"] != 0.0:
                        cX = int((M["m10"] / M["m00"]))
                        cY = int((M["m01"] / M["m00"]))
                        cv2.circle(image, (cX, cY), 7, (0, 0, 255)
                            ↪ , -1)
                # draw the contours on the image
                c = c.astype("int")
                cv2.drawContours(image, [c], -1, (0, 0, 255), 2)

        cv2.imshow("Image", image)
        cv2.waitKey(1)
```

Listing 4: Calculate x-position of recognized charging station

Some preprocessing is performed to the extracted pixels in step 03 before the midpoint will be calculated (see Listing 4). Preprocessing reduces noise and makes the data more reliable. For preprocessing, the image is grayscaled first using the *cv2.cvtColor()*-function of OpenCV [14]. Next step is to blur the image using the *cv2.GaussianBlur()*-function and then threshold the blurred image to drop all pixels which have a value lower 130 using the *cv2.threshold()*-function. For more details see [14].

In step 04 different OpenCV functions are used to find contours in the thresholded image (see Listing 4). The function *cv2.findContours()* finds ”the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition” [14]. For more information see [14].
The found and returned contours and hierarchy are stored in the variables *cnts* and *hierarchy*. The variable *hierarchy* is currently not further used.
Next, the variable c is iterated and represents the found contours in each iteration. The moments for one contour, which include information about the weighted average of image pixel intensities, are stored and can be accessed in variable *M*. By accessing attributes of variable M, it is possible to calculate the x- and y-coordinate for the center of the respective contour [14].
Using the *cv2.circle()* and *cv2.drawContours()* functions, the found and calculated contours and the gravity-center of each contour are added to the originally captured image. Figure 14 shows the result of the implemented and described OpenCV-functions.

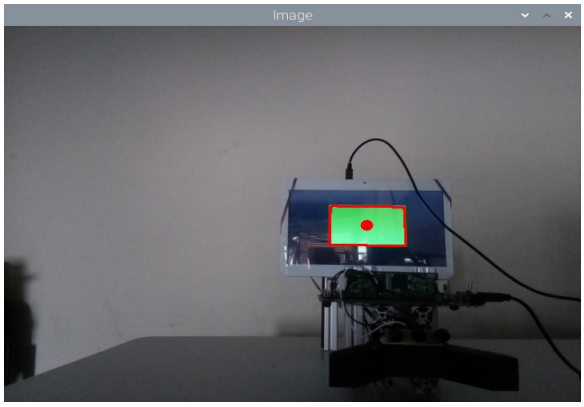


Fig. 14: Detected contour and midpoint

The cX-value provides the horizontal location of the charging-station-center. With an image-width of 640 pixel, it is possible to calculate wether the charging-station-center is in the right, in the left or in the middle of the picture by

applying following substraction:
xValue = cX-320.
The detected and calculated value is now in the range of -320 to 320. The Raspberry Pi preprocesses the calculated x-value of the mid-point of the detected charging station by multiplying it with the factor 255/320. The gained value in the range of -255 to 255 is then sent to the CAN-Bus as a message with CAN-ID 0x103. Sending a CAN-message with the ID 0x103 to the CAN-Bus implies that a charging station was detected.

Since the visual detection is used for the wide-range detection of the charging station, the maximum distance that still provides a reliable detection has been investigated and results in 6 m in a dark environment without overhead lighting. The maximum distance in an environment with overhead lighting is 4 m.

B. CAN Communication

The PCBs of the vehicle communicate via CAN-messages among each other.
The x-value of the detected station as well as some other sensor-information provided on the CAN-Bus allow the implementation of an algorithm for autonomous driving behaviour of the car. The messages provided by the Raspberry Pi and the sensor PCB as well as externally sent CAN-AktDriveSet- and AktSteerSet-messages are read by the actor-PCB and corresponding to the data sent with these messages, the driving and steering actions are performed. Table II shows the CAN-messages and their Identifier used for the communication in this research-project.

Identifier	**Name**	**Description**
0x00	NOTHALT	Emergency Stop
0x38	SensDistanceUs	US Distance front and back
0x52	AktStop	Stops the actors
0x53	AktDriveSet	Contains parameters for drive-motor
0x55	AktSteerSet	Contains parameters for steering-motor
0x99	AutonomModeOff	Turn Off Autonomous Mode
0x100	AutonomMode Toggle	Turn On Autonomous Mode
0x103	ChargingStation Detected	Turn On Wide_Range Approaching Mode
0x104	DockingStation Detected	Turn On Close_Range Approaching Mode

TABLE II: CAN-Message Overview

The hardware and software implementation of the near-field docking process using the IR-LEDs and the photodiode is based on the work of Philip Meißner (see [5]). In his work, Meißner implemented CAN-messages with identifier 0x101 which is sent from the docking station as a docking ”heartbeat”. When the docking and charging process is completed, the docking-system sends a CAN-message with ID 0x102 to the CAN-Bus [5]. Implementing the communication to the docking station will not be part of this research-project, as the main focus of this project was the procedure for approaching the docking station. Therefore the CAN-message with ID 0x103, which provides information about the location of the charging-station for the wide range docking process and the CAN-message with the ID 0x104, which provides information about the location of the charging-station using the photodiode for the close range docking process are implemented.

VI. Realization of docking process

As there are different steps for the vehicle to approach the charging station in the correct orientation, the finite state machine, which is implemented on the actor-PCB-microcontroller TMS320F28335, will be split up to ease its understanding. Furthermore the output-vector [DriveMotor, SteerMotor] is simplified. 1 means, that the respective motor performs an action. 0 means, that the motor is not actuated. Figure 15 shows the basic implementation of the manual mode, which is activated when the vehicle starts. Steering and driving actions are only performed, when the respective messages are sent to the CAN-Bus from the Raspberry Pi or from an external participant connected with a cable. By sending a CAN-Message with the ID 0x100, the autonomous mode of the vehicle is activated and the vehicle begins a sequence of steps for finding a charging station (see Figure 16). As shown in Figure 16, the sequence is just drive straight forward, until there is no more space in front of the vehicle and then drive straight backwards, until there is no more space in the back of the vehicle. This routine for finding a charging station is considered to be sufficient in this project.

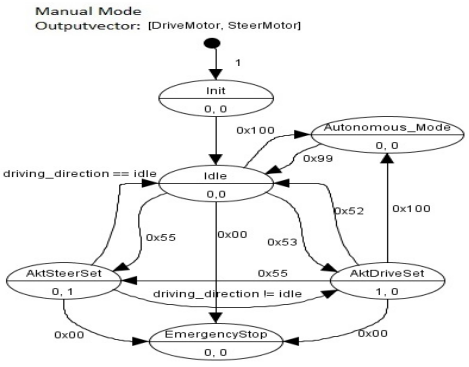


Fig. 15: StateMachine Manual Mode

A. Wide range

As soon as a charging station is detected, the Raspberry Pi provides the location of the charging station by sending a message with Identifier 0x103. When the vehicle is in

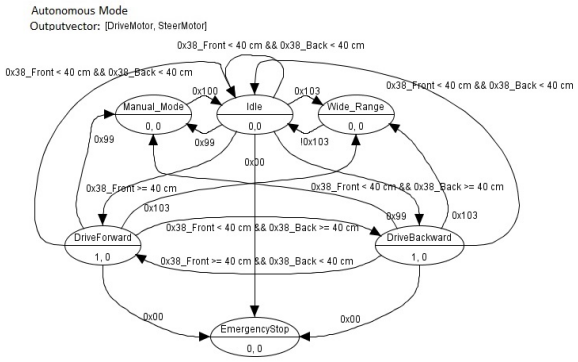


Fig. 16: StateMachine Autonomous Mode

Autonomous Mode and a CAN-message with ID 0x103 is received, the routine for the wide range docking process will be started.

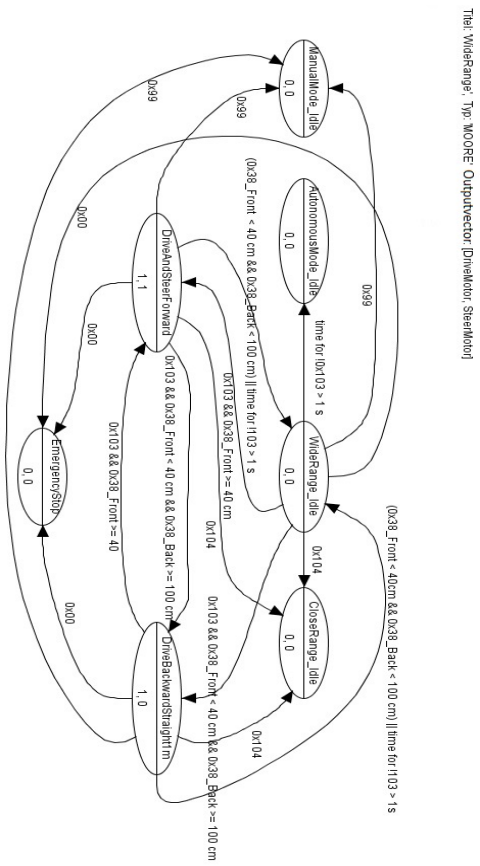


Fig. 17: StateMachine Autonomous Wide Range Mode

Therefore, the steering angle will be adjusted according to the x-value of the charging station continuously (see Figure 17). Also the vehicle lowers its speed and checks the distance in the front. When the distance is less than 40 cm and there is no sufficient signal from the IR-LEDs to perform the close range docking process, which would be indicated by a message with identifier 0x104 from the sensor-PCB, the vehicle moves back for 1 meter with straight steering direction while moving backwards (see Figure 17). Then the vehicle starts to approach the charging station again. When the front-distance of less than 40 cm is reached and there is sufficient signal from the IR-LEDs, the close range docking process will start. The information about the IR-LED-signals are provided in the CAN-message with ID 0x104 from the sensor-PCB (see Figure 17).

When there is no CAN-message with Identifier 0x103 received for more than one second, the vehicle switches back from wide range docking process to autonomous mode and processes steps for finding a charging station (see Figure 16).

B. Close range

Since the wide range approach was implemented successfully and ends when there is sufficient signal from the IR-LEDs, the vehicle just stops at this point and the close range approach is to be continued in another project. While testing, it was noticable that there were some issues with the alignment of the IR-LEDs at the charging-station. It should be considered to design a PCB for the docking station to make the system reliable and insensitive towards touching and unintended bending of the IR-LEDs which leads to deviating pulse-information at the photodiode. Furthermore, a PCB helps to reduce the amount and length of the wires. This could improve the safety while charging the battery and ensures a sound CAN-communication amongst the charging-station and the vehicle.

VII. Existing problems and difficulties

While developing the manual and autonomous driving approach of the vehicle, it has been jacked up on a table to prevent it from driving. The testing and evaluation of the behavior of steering and driving motor was successful. The final tests were made in an actual environment of the vehicle without jacking it up. Unfortunately while driving on the floor, problems with the steering of the vehicle occured. The front of the vehicle is too heavy and the weight in addition with the rubber material of the wheels prevent the wheels from showing expected steering behaviour on the ground. Especially when starting in straight mode, the first steering action to the left or right is performed fine, but going back to straight in manual mode can evoke faulty steering angles, although the PWM signals are sent to the motor-driver. Since the PWM signals are sent, the PWM interrupts, which control the inner counter of the microcontroller to calculate the steering direction, occur. So the calculated steering direction and the actual steering direction of the car do not correspond. To correct this misbehaviour, the mid-part of the vehicle has been extended to distribute the weight and to relieve the front axis. Furthermore, a one Ampere steppermotor was substituted with

a three Ampere steppermotor. To ensure the driving belt to be adjusted solid without any margin, a 3D-guide role was printed and integrated in the vehicle. Nevertheless, the weight of the car should be reduced as much as possible in the future and the steering mechanism needs to be overhauled. Indeed, a slide potentiometer to provide feedback about the steering angle is already built in, but the ADC connection of the sensor-pcb is used to connect the photodiode for close-range-detection. So when implementing the delevoped system permanently, the sensor-pcb as well as the steering mechanism need to be overhauled.

Another difficulty noticed while testing was the reflection of light at the surface of the tablet, which is used to display the visual landmark. Overhead light or a bright environment in combination with an inconvenient angle of the tablet could lead to a superposition of the environment light and the green rectangle, which is used as visual landmark. To prevent this, anti-mirrorfoil could be used. In this project, the vehicle was tested in a dark environment, which results in a good recognition of the visual landmark from up to 5 m. When running the system in different environments with different lighting conditions, it is possible, that the rectangle shown on the display does not match the predefined HSV-value anymore. So for different environments, the lower and upper HSV threshold has to be checked and matched for the system to show desired behaviour.

In addition to that, the visual landmark is not recognizable anymore when the angle of view is lower than 30° (see red areas in Figure 18). An approach to get a wider angle of

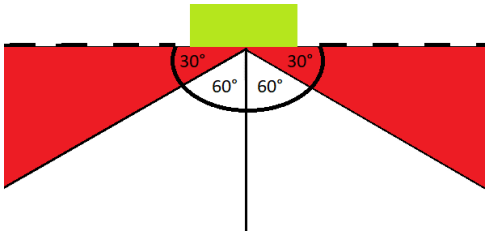


Fig. 18: Recognizability of visual landmark depending on angle

view is to use an orange DINA4-sheet as visual landmark. For this approach, several images of the sheet were taken and the max and min BGR values were evaluated using python and OpenCV.

Checking all pixel-values of the taken images of the orange sheet results in max and min BGR-values for the orange sheet. The min and max RGB-values are then used to find the respective boundaries in the HSV-space using an online rgb-hsv converter (see Figure 20).

Since OpenCV uses hue range [0,179], saturation range [0,255], and value range [0,255] for HSV [14], manual conversion has been done to these boundaries.

Currently, this approach does not improve the recognition of the visual landmark. However this approach should be

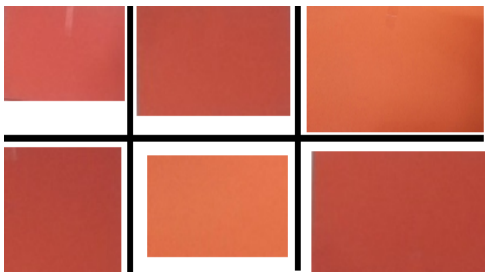


Fig. 19: Taken images for pixelvalues of orange sheet

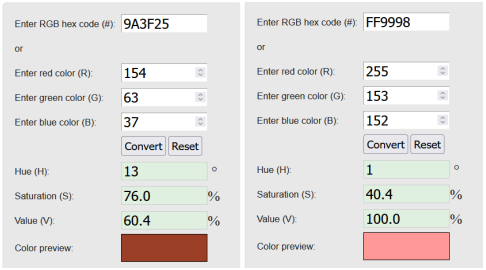


Fig. 20: RGB to HSV conversion

investigated and refined in the future.

The combination of the imageprocessing-algorithm and the already implemented ROS could be an additional improvement of the whole system. The image processing was developed in python and needs to be started manually at every start of the RosPi-system. If the python-program for recognition of the landmark is not started manually, the algorithm does not provide information about landmark recognition on the CAN-Bus. Implementing the python algorithm into the ROS-system in C++ or starting the python-imageprocessing-process autonomously while booting would be a more neat solution. The imageprocessing-information would then be sent to the CAN-Bus via CAN-accessories of ROS and information about a detected charging station would be available on the CAN-Bus with the start of RosPi and without a manual action of the user.

Furthermore, the ROS implementation includes code to convert and forward signals coming from an XBox-Controller to the CAN-Bus. Since the implementation of this recent project includes a possibility to switch between manual and autonomous mode, a currently not used button of the XBox-Controller could be determined to switch between both modes. The manual mode works fine with the XBox-Controller. A possibility to switch to autonomous mode and back with the Remote Controller would be an enjoyable feature for the user.

VIII. Results and Conclusion

The autonomous wide-range approach performed by the vehicle based on the taken images and image-processing shows

desired behaviour when starting in a central position with convenient alignment towards the charging station.

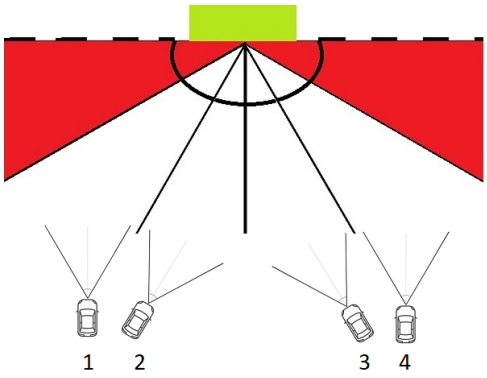


Fig. 21: Different starting positions for wide-range approach

When starting in a placement as shown in position 1 or 4 in Figure 21, the vehicle approaches the station but cannot recognize the infrared-pulses. So it ends up in a loop of going forth and back (see Figure 17). Starting in a placement between position 2 and 3 (see Figure 21) with alignment towards the station, the vehicle ends up in a position from which it is possible to recognize the infrared-led pulses and to gain information about its close-range orientation towards the charging station. Additional movements could then be initiated to get to the right position for battery-charge.
Conclusively, a reliable system for wide-range detection and approach of a charging station has been developed. The framerate of the Raspberry Pi camera in combination with the information extracted by image processing is sufficient to perform an approach towards the charging station autonomously from up to 5 meters. Via the wide-range docking process, the vehicle approaches the docking station autonomously and performs central positioning dependent on the starting position. Approaching the center of the charging station, the vehicle stops when the signals of the IR-LEDs are sufficient. The IR-LED-signals could then be used to perform the close-range docking process. The close-range system developed by Alexander Meißner is able to provide the necessary information for up to 61 cm and to finalize the docking process [5]. Merely the hardware and the setup of the charging station should be revised by building a PCB for the system and prevent the IR-LEDs from being moved. The algorithm for the close range docking process needs to be implemented as well.
Furthermore the camera provides images of the environment. This can be useful in future works, especially works that involve mapping spaces and different types of object recognition. Different objects could be equipped with different visual landmarks. The image processing code needs to be extended with shape- or color-detection then. Furthermore sensor information, for example a laser communication with a charging station, could be matched with the information extracted from the images.
The most important aspect for the desired functionality of a reliable wide-range approach is a reliable steering mechanism. Misalignment caused by mismatching of mechanics and sensor information needs to be prevented. Furthermore reflection of environment light should be evaluated and reduced as much as possible while searching for the site location of the charging station. When these improvements were made, the vehicle presents a solid system for manual and autonomous driving actions.

IX. Acknowledgment

The realization of this project took place in the “Embedded Communications and Signal Processing Lab” which is part fo the “Institute of Communications”. The “Institute of Communications” appertains to the “Faculty of Information, Media and Eletrical Engineering” of the TH Köln - Cologne University of Applied Sciences. The author expresses special thanks to Prof. Dr. Bartz for the opportunity to implement this project in the laboratory under supervision and support of Prof. Dr. Bartz.

References

[1] Deloitte, “Urbane Mobilität und autonomes Fahren im Jahr 2035”, Online: https://www2.deloitte.com/de/de/pages/trends/urbane-mobilitaet-autonomes-fahren-2035.html, Last reviewed: January 2022.
[2] Verband der Automobilinsdustrie, “Erstes globales E-Mobility-Ranking”, Online: https://www.vda.de/de/presse/Pressemeldungen/210423-Erstes-globales-E-Mobility-Ranking, Last reviewed: January 2022.
[3] Robert Rose, “Entwurf und Implementierung einer skalierbaren Kommunikation zwischen mobilen Systemen unter Verwendung von ROS”, Bachelor-Thesis TH Cologne: 2019.
[4] Robert Rose, “Development of a Platform-Independent CAN Monitor and Parametrization Tool for Specific Devices”, Research Project TH Cologne: 2020.
[5] Philip Meißner, “Entwicklung eines Docking-Systems zum autonomen Laden eines RC-Modellfahrzeugs”, Master-Thesis TH Cologne: 2019.
[6] Raspberry Pi (Trading) Ltd., “Raspberry Pi 4 Model B Datasheet”, Online: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf, Last reviewed: January 2022.
[7] NVIDIA CORPORATION, “JETSON NANO DEVELOPER KIT”, Online: https://cdn-reichelt.de/documents/datenblatt/I100/NVIDIA_JETSON_NANO_ENG_MAN.pdf, Last reviewed: January 2022.
[8] Raspberry Pi (Trading) Ltd., “Raspberry Pi Camera Module”, Online: https://www.raspberrypi.com/documentation/accessories/camera.html, Last reviewed: January 2022.
[9] KTelectronic, “Ultraschall Messmodul HC-SR04”, Online: https://www.mikrocontroller.net/attachment/218122/HC-SR04_ultraschallmodul_beschreibung_3.pdf, Last reviewed: January 2022.
[10] Sparkfun, “Ultrasonic Ranging Module HC - SR04”, Online: https://cdn.sparkfun.com/assets/b/3/0/b/a/DGCH-RED_datasheet.pdf, Last reviewed: January 2022.
[11] “MD03 - 24Volt 20Amp H Bridge Motor Drive”, Online: http://www.robot-electronics.co.uk/htm/md03tech.htm, Last reviewed: January 2022.
[12] Adrian Rosebrock, “OpenCV shape detection”, Online: https://pyimagesearch.com/2016/02/08/opencv-shape-detection/, Last reviewed: January 2022.
[13] “HSV-Farbraum”, Online: https://de.wikipedia.org/wiki/HSV-Farbraum, Last reviewed: January 2022.
[14] OpenCV, “OpenCV modules”, Online: https://docs.opencv.org/4.x/, Last reviewed: January 2022.

Development of a Robust Peak Detection Algorithm in Time Series Data

Shezan Hossain Mahmud
Communication Systems and Networks
TH Köln
Köln, Germany
shezan_hossain.mahmud@smail.th-koeln.de

Ender Akcöltekin
Backbone Portfolio Development
Varian Medical Systems (A Siemens Healthineers Company)
Troisdorf, Germany
ender.akcoeltekin@varian.com

Dr. Cyrano Bergmann
Software Engineering
Varian Medical Systems (A Siemens Healthineers Company)
Troisdorf, Germany
cyrano.bergmann@varian.com

***Abstract*— Common practice in sensor data processing is to detect measured peak or trough signals in trend charts retrieved from a machine or a system. Those signals are commonly signs of anomalies or meaningful events. The goal of this paper is to present a peak or trough detection routine, primarily to find all valid peak or trough signals in historical sensor data of the system condition in a medical cyclotron and decide which of those could be relevant for further analysis. Peaks or troughs with e.g., sudden sharp rise or fall in value can be analyzed, classified, and eventually correlated with the potential cause of their occurrence, such as component lifetime, malfunction, exceeding system limits, etc. The particular focus of this work comprises the development of a concept and the implementation of an algorithm that is capable to detect peak signals automatically in time-dependent cyclotron sensor trend data with minimal predetermined threshold information. The results of this work create fundamentals for the development of predictive maintenance mechanisms.**

Keywords— Cyclotron sensor data, Time-series data, Peak Detection, Trough Detection

I. Introduction And ProbLem Statement

The cyclotron considered here is a type of circular accelerator for protons (nucleus of hydrogen atoms) and as such, it is responsible to accelerate protons up to roughly 60% of the speed of light and extract them as a continuous beam of protons for irradiation treatments of cancerous tumors. This type of treatment is widely known as Proton Therapy (PT). Varian Medical Systems PT (Varian PT) is one of the world's leading suppliers of PT system solutions and has built more than two dozen treatment centers worldwide. Therefore, Varian PT is committed to ensuring high system quality, reliability, and stability in clinical operations. To ensure these goals, continuous monitoring and controlling of the overall system condition at any time is mandatory. The cyclotron is a very complex system including thousands of sensors. These sensors generate continuously millions of data points within a short period. It is very time-consuming to analyze these data manually to understand the anomalies or meaningful events. An automatic robust peak detection algorithm can certainly pave the way to easily analyze the data and find out interesting correlations among them. Oftentimes this type of data study aids in anticipating system failure and identifying it before it occurs. Not only that, in the case of remote services of the cyclotron, automatic peak recognition can be a very effective way to comprehend the peaks of different sensors' data and their relationships.

Although it seems easy to detect manually a peak or trough signal in a small time series, dealing with a large amount of data and detecting peaks automatically and reliably, increases the challenge to develop an algorithm that is capable to find all relevant peak signals from the data within a short period, as the false peaks detection incorporates with true peaks. Furthermore, distinguishing the noise and peak level without any predetermined threshold or influence increases the difficulty of automatic robust peak detection in time series data. All local peaks are not necessarily true peaks. True peak values need to be so large that even in a global context they appear as significantly large values [1].

II. Related Works

Some works on peak detection have been evaluated. By separating noisy pixels from noise-free pixels, the appropriate pixels replaced the noisy pixels during the filtering stages [2]. Various threshold values have been applied to achieve this. These cutoff points were derived using the mean, standard deviation, and quartile. With an auxiliary condition to identify noisy pixels, these three threshold values were applied one after the other. This method is very interesting for detecting noise from noise-free pixels. Another research proposes several alternative methods for peak detection and compares them using annual sunspot data [1]. For improved performance, the Automatic Multiscale Based Peak Detection (AMPD) algorithm's standard deviation was replaced with variance [3]. The strongest peaks are identified using moving mean and the standard deviation of the moving mean before comparing the overlaps of burst areas found in the time series [4]. These algorithms cannot be adapted directly, but new methods for cyclotron sensor data are required.

III. Background And Required Tools

The new algorithm defines all the entries with a larger distance to the mean as outliers. The dataset chosen to illustrate the peak detection performance of the algorithm is a vacuum sensor used to measure the pressure conditions in the cyclotron's ion source, which is operated in a pressure range between 1E-04 mbar and 1E-06 mbar, but the system pressure might reach up to 100 mbar. Operation of the cyclotron includes frequent machine maintenance and other user operations that directly affect the pressure in the cyclotron, such as purging and venting operations. As a result, unstable conditions are visible, which results in a moving or drifting baseline. This leads to a complex scenario in the global distribution of the full dataset (fig: 1).

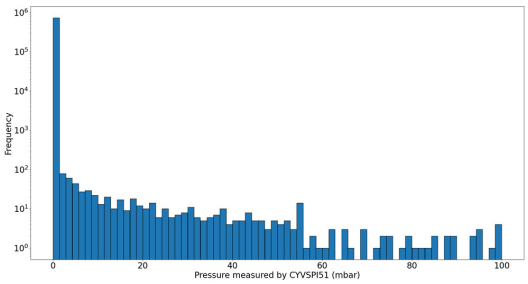


Fig. 1: Histogram of the measurements of vacuum sensor CYVSPI51

It is expected that the baseline signals, which come from the preceding purging and venting events, follow a Gaussian distribution and should be recognized above the stable operating conditions. Figure 2 shows the situation from a close-up of the sample values, where an underlying Gaussian distribution can be seen, transitioning to an exponential decay toward higher values. This agrees well with the assumption that venting a fixed column with constant power pumps will result in an exponential pressure drop in the column after venting. Therefore, it is expected to find fewer measurements with high-pressure values than with low pressure values.

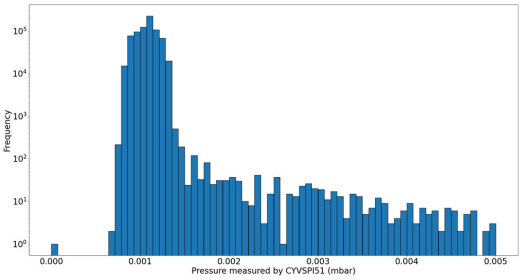


Fig. 2: Histogram of the measurements of vacuum sensor CYVSPI51 (zoomed towards maximum frequency)

IV. Tools Required

Notebook (16GB RAM), Windows 10, Notepad++, Jupyter Notebook, Python 3.8.

Python Libraries: Pandas, NumPy, Matplotlib, SciPy, seaborn, scikit-learn.

V. Algorithm Analysis

The algorithm works in the way that it detects values out of the range of the n^{th} standard deviation around the mean. Since the global conditions in the cyclotron vacuum are unstable, the algorithm utilizes the local stability within a moving window to calculate a local threshold to detect outliers in the dataset. The iterative approach is utilized to enhance the robustness and sensitivity of the algorithm. The detected peaks (outliers) are separated from the dataset at the end of each loop, which lowers the threshold values towards the baseline after each loop. This iterative approach also increases sensitivity for smaller amplitude peaks.

A. Algorithm for Peak Detection

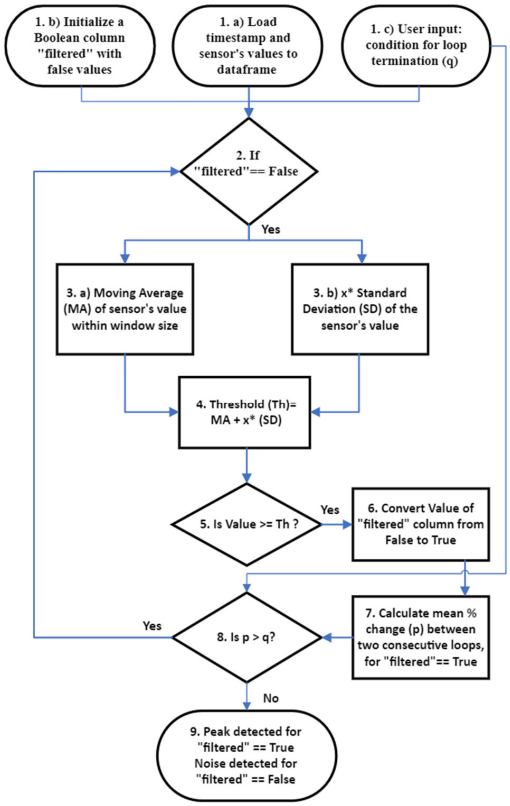


Fig. 3: Algorithm for peak detection

Figure 3 represents the steps of the algorithm for peak detection. The method of calculating "cut-off using moving mean and n*standard deviation", used in [4], is modified to utilize in this project to detect peak signals. The data has been collected from the log files of Cyclotron and copied as a data frame of timestamp. Each step of the algorithm is briefly illustrated below:

1. a) Timestamp (t_i) and sensor values (v_i) are loaded to data frame.

   b) A Boolean column "filtered" is initialized with all False values.

   c) To terminate the loop of peak and noise separation, a numerical user input (q) is also initialized.

2. Data frame with corresponding False values in "filtered" column are forwarded for further calculations.

3. a) The moving average (MA) of the sensor values within a suitable window size is calculated.

   b) Also, the x^{th} standard deviation of the dataset is calculated. (Here, x is a real variable and needs to be adjusted according to dataset's characteristics).

4. The threshold (Th) is calculated by the summation of the calculated MA and x^{th} standard deviation.

5. A comparison is made between the sensor's value and the calculated threshold (Th).
6. If the sensor's value is greater than or equal to the threshold, the corresponding "filtered" column's Boolean False value turns into True, otherwise remains as False.
7. The mean percentage change (p) between two consecutive loops of the sensor's value is calculated for "filtered" == True.
8. If mean percentage change (p) is lesser than the defined q value, the loop is terminated, otherwise it is continued executing from step 2.
9. Peak signals are detected with mask "filtered" == True. And noise values are recognized with mask "filtered" == False.

B. *Algorithm for Trough Detection*

The algorithm of peak detection described above is slightly modified to recognize the trough signals. The modified steps are briefly mentioned following:

- In step 4, instead of summing MA and x^{th} SD, absolute difference between them is used to define the threshold.
- The "greater than" sign is altered to the "less than" sign of step 5 to detect troughs.
- Since mean percentage change is less significant for trough values, for terminating iterations, the count percentage change is used in step 7 instead of the calculating mean percentage change.

VI. Evaluation of Peak Detection

A. *Result Demonstration of New Algorithm*

To demonstrate the result of the algorithm, the dataset of vacuum sensors (CYVSPI51) has been selected. The following plot (fig: 4) shows the dataset representation before applying the algorithm, where the x-axis represents the timestamp in the second (from January to March) and the y-axis represents the pressure measurements of the corresponding timestamp in millibar.

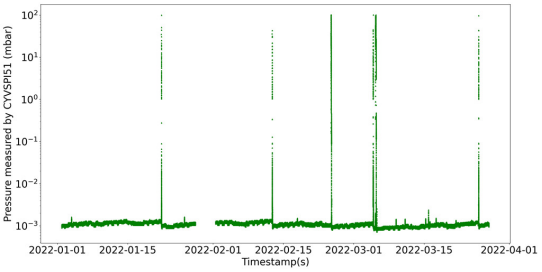


Fig. 4: Pressure measurement by sensor "CYVSPI51" for 3 months (January to March 2022)

The outcome of the new algorithm after performing 6th iterations is plotted following (fig: 5), where the red stars represent the detected peak signals, and the cyan dots correspond to noise values in millibar. For this dataset, to calculate the thresholds, the summation of the moving mean (within a window size of 1000) and 3rd standard deviation has been selected.

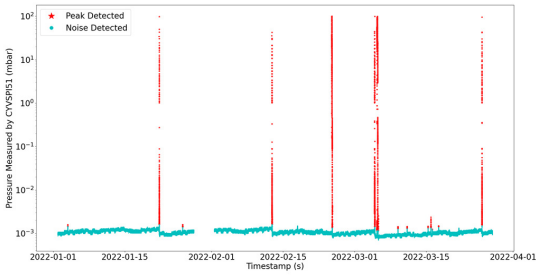


Fig. 5: Plot of peak pressure detected by the new algorithm after the final (6th) iteration

After the final iteration, the new algorithm recognized peak signals in the pressure range between 1E02 mbar and 2E-03 mbar (approximately). The following zoomed version of the same figure shows better visualization of the plot to understand.

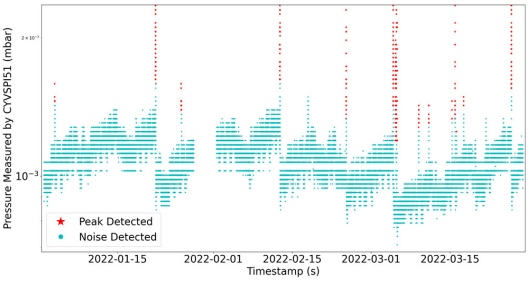


Fig. 6: Plot of peak pressure detected by the new algorithm after final (6th) iteration (zoomed towards noise level)

It can be noticed from figure 6 that, the new algorithm detects no peak (red stars) at the noise level, which means no false peak is recognized. The algorithm needs only 27.22 seconds to find such peaks among the 730505 vacuum sensor data points and stores the figures for all the loops in the local system.

B. *Comparison of the New Algorithm With DBSCAN*

To test the accuracy of the new algorithm a comparison has been made using the DBSCAN (Density-based Spatial Clustering of Applications with Noise) algorithm [5] of the scikit-learn library on the same dataset. It is needed to be noted that the detailed description of the DBSCAN algorithm and its parameter optimization technique is out of the scope of this paper. The following plot (fig: 7) shows the outcome of the DBSCAN algorithm on the same vacuum sensor dataset.

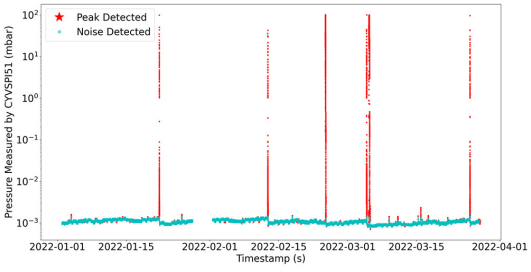


Fig. 7: Peak and noise pressure separated by the DBSCAN algorithm

Figure 7 illustrates peaks (red stars), and noise (cyan dots) detected by DBSCAN. As seen in the picture, some of the erroneous peaks that DBSCAN identified are actually in the noise level. The following figure (fig: 8) is the zoomed version of figure 7 shows the highlighted false peaks (red stars at the noise level) recognized by DBSCAN.

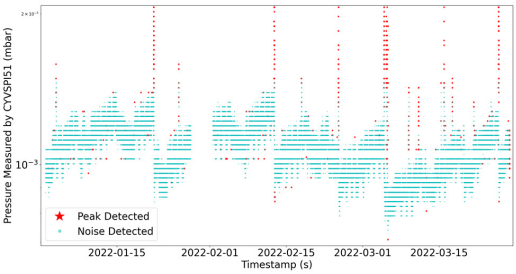


Fig. 8: Peak and noise pressure separated by the DBSCAN algorithm (zoomed at the noise level)

The next plot (fig: 9) shows a direct comparison between the two algorithms and demonstrates where DBSCAN incorrectly recognized some noise that the new algorithm correctly detected as peak signals (red stars).

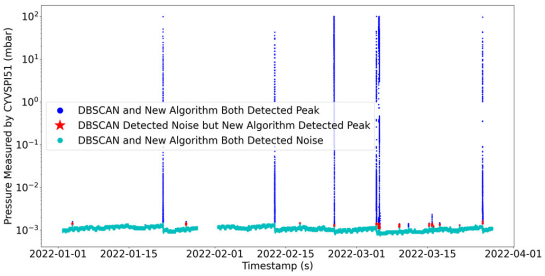


Fig. 9: False noise detected by DBSCAN, new algorithm recognized those correctly as peak signals

In January and March, the false noises (red stars) of figure 9 are more noticeable. The following zoomed variants of figure 9 provide a vivid illustration of the scenario stated above for those months.

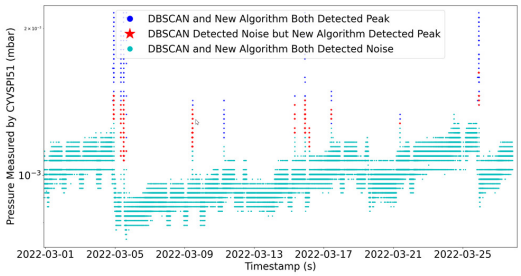


Fig. 10: False noise detected by DBSCAN, new algorithm recognized those correctly as peak signals (zoomed in the month of March)

Figure 10 displays some false noise (red stars) detected by DBSCAN in March. While DBSCAN detected those values (red stars) as noise, it can be comprehended that these values are peak signals, which the new algorithm recognized correctly. The same scenario is also noticeable in January and illustrated by figure 11.

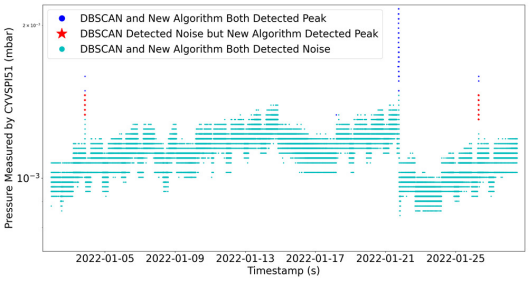


Fig. 11: False noise detected by DBSCAN, new algorithm recognized those correctly as peak signals (zoomed in the month of January)

Additionally, DBSCAN detected a few erroneous peak signals at the noise level (fig: 12), whereas the new algorithm properly identified those values as noise.

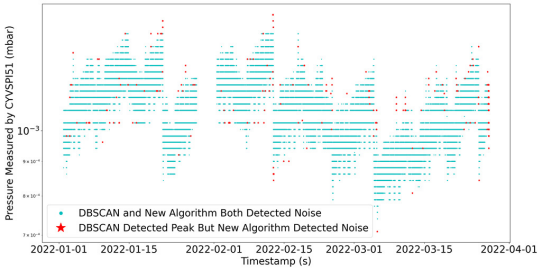


Fig. 12: False peak signals detected by DBSCAN; new algorithm recognized those correctly as noise (zoomed at the noise level)

In a nutshell, according to the comparison graphs described above, the new algorithm is recognizing fewer erroneous peak signals and noise values than the DBSCAN algorithm for the vacuum sensor dataset.

C. Accuracy Calculation

The number of peak signals has been manually calculated within the period of 1st January to 15th February, to determine which method is demonstrating better accuracy. From 349106 data points, 650 data (approximately) were selected as peak signals by manual calculation. The following accuracy measurements have been obtained using the manual peak calculation as the reference.

Method	True Positive (%)	True Negative (%)	False Positive (%)	False Negative (%)
New algorithm	95.08	100	0	4.92
DBSCAN	83.85	99.95	0.05	16.15

Table 1: Accuracy calculation using manual calculation as reference

It is needed to be noted that, manual peak calculation is very time-consuming, and it may also inject some amounts of human errors. Based on the comparisons made above, it can be said that the new algorithm performs better at detecting

true peaks and true noise while having lower false positive and false negative percentages than the well-known DBSCAN.

D. Result Demonstration for Trough Detection

The upper pole cap's water temperature measurement sensor dataset of cyclotron is used to explain the outcome of trough detection. The plot below (fig: 13) represents the six months of data of the sensor before applying the new algorithm.

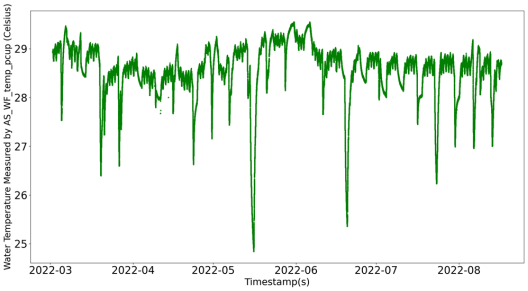


Fig.13: Temperature measurement by sensor "AS_WT_temp_pcup" for 6 months (March to August 2022)

The following plot (fig: 14) displays the result of the final iteration of the algorithm for trough detection, the red stars signifying the identified troughs and the cyan dots corresponding to the noise values. The absolute difference between the moving mean (within a window size of 1000) and the 2nd standard deviation is chosen for this dataset to compute the thresholds.

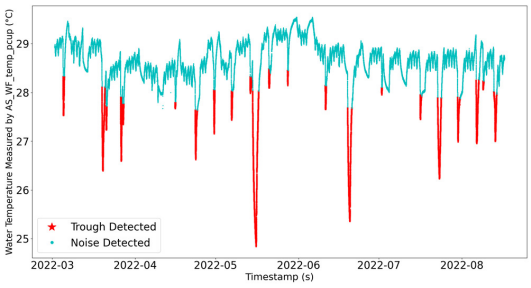


Fig. 14: Plot of trough and noise temperature detected by the new algorithm after the last (16^{th}) iteration

After the final iteration, the new algorithm recognized trough signals in the temperature range between 24.8°C and 28.5°C (approximately). The algorithm takes only 35 seconds to separate troughs from 241793 data points of the temperature sensor and store the figures of each iteration in the local system.

On the other hand, the parameters of the DBSCAN algorithm could not be adjusted similarly compared to that used for peak identification, since the time difference between two successive data stored (Δt) is approximately constant for all the points of this temperature sensor's dataset. Thus, the flexibility of the new algorithm is demonstrated by its ability to detect troughs up to a satisfactory level.

VII. Dependencies of the Algorithm

The new algorithm for peak and trough detection has some dependencies which are discussed below:

A. Window Size of Moving Average:

The window size affects how efficiently the new algorithm works. Sticking to a specific window size is challenging because each dataset of cyclone sensors has a unique property and a variable value range. As a result, the window size is a variable algorithmic parameter.

If the window size is selected too small, the thresholds will have sharp edges, which will lead to some false peak identification. On the contrary, selecting a bigger window size can enhance the inability to recognize some true peak signals. So, this parameter needs to be adjusted according to the dataset's characteristics.

B. Multiplication Factor of Standard Deviation:

Another arbitrary parameter of the new algorithm is the multiplication factor of standard deviation. If this parameter is chosen too small, there will be false peaks incorporated with true peak signals. Again, fewer true peaks or troughs will be recognized as the multiplication factor of standard deviation increases. Depending on the dataset's properties, the multiplication factor of standard deviation needs to be adjusted.

C. Condition for Loop Termination:

The mean percentage change of selected peaks between two successive loops has been chosen to terminate the loop for peak detection. Again, to end the iterations for trough recognition, the percentage change of the number of selected troughs between two consecutive loops is selected in the new algorithm. Depending on the characteristics of the dataset, this parameter may need to modify.

VIII. Conclusion

A new algorithm based on the statistical dispersion method has been developed to detect peak or trough signals as well as noise in sensor data of Varian's cyclotron. To limit the number of data, peak signals have been analyzed only on the vacuum pressure data, whereas troughs signals were analyzed on temperature data measured in the cyclotron's pole caps. An important advantage of this algorithm is the speed of signal detection which is enabled by a masking technique that prevents the creation of multiple data frames during the iterative approximation.

Furthermore, using manual peak and noise analysis as a benchmark, the new algorithm is showing better accuracy in recognizing true positive ($\approx$95%) and true negative (100%) than the DBSCAN algorithm in the dataset of vacuum pressure. Again, in contrast to the DBSCAN algorithm, the new algorithm also exhibits fewer false positive (0%) and false negative ($\approx$ 5%), respectively.

Additionally, those efficiency calculation statements are also supported by direct comparison plots between two algorithms, where it can be comprehended that some erroneous peaks and noise detected by DBSCAN were correctly identified by the new algorithm. Moreover, by analyzing the dependencies of the new algorithm, peak or trough signals can be detected up to a satisfactory level using this algorithm.

Identified peaks or troughs from these analyses of cyclotron sensor data time series are data points that require further investigation. E.g., step functions where measured vacuum pressure reaches a new level will be found with high confidence, but whether the new level indicates some maintenance requirements or can be described by some other physical effects is currently still part of deep expert analysis.

The number of the analyzed dataset has been limited by the demanded resources for reference data analysis. And further analysis of different datasets is postponed for later projects.

IX. Acknowledgement

Special thanks to Prof. Dr. Andreas Grebe, the supervisor from TH Köln, for guiding the project with his vast experience and expertise.

X. References

[1] G. Palshikar et al., “Simple algorithms for peak detection in time-series,” in Proc. 1st Int. Conf. Advanced Data Analysis, Business Analytics and Intelligence, vol. 122, 2009

[2] N. Singh and U. Oorkavalan, “Triple threshold statistical detection filter for removing high density random-valued impulse noise in images,” EURASIP Journal on Image and Video Processing, vol. 2018, no. 1, pp. 1–16, 2018.

[3] A. M. Colak, Y. Shibata, and F. Kurokawa, “Fpga implementation of the automatic multiscale based peak detection for real-time signal analysis on renewable energy systems,” in 2016 IEEE International Conference on Renewable Energy Research and Applications (ICRERA). IEEE, 2016, pp. 379–384.

[4] B. Rossi, B. Russo, and G. Succi, “Analysis of open source software development iterations by means of burst detection techniques,” in IFIP International Conference on Open-Source Systems. Springer, 2009, pp. 83–93.

[5] GeeksforGeeks, “DBSCAN Clustering in ML — Density based clustering,” https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering, 24 August 2022