Felix Paper Title*

Felix Girke
Computer Science and Engineering
Frankfurt University of Applied Science
Frankfurt, Germany
https://orcid.org/0009-0004-3967-6750

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—To solve the problem of different interfaces for different robots a data connector is developed. It creates a interoperable, decentralized Network for a robot system using the Open Platform Communication Unified Architecture (OPC-UA) standard. Creating a flexible digital twin in Isaac Sim to visualize the data of the robot system. Remote access to the data connectors (and the digital twin) to monitor the robot system from everywhere.**

***Index Terms*—OPC-UA, interoperable, decentralized, digital Twin, Isaac Sim, remote access**

I. Introduction

Every robot manufacturer develops uses their own way of communication with their robot. This leads to problems when trying to build a robot system with multiple robots from different brands. In this example two robot arms from Kinova are to be mounted on a Husky mobile robot platform from Clearpath. The Husky uses Robot Operating System (ROS) to communicate internally while the robot arms use the Kortex api from Kinova. To combine them into one digital twin they should be on one standard. For this a data connector is developed on the Open Platform Communication Unified Architecture (OPC-UA) standard. Its purpose is to act as a layer between the robot specific language and the outside world, in this example the digital twin. To add flexibility the robot data isn't collected on one server but every robot has its own server. Because of this decentralized approach every client can choose to connect only to the servers it wants to. For example if only one of the robot arms is mounted on the husky the digital twin can connect to only this one while the other robot arm con be used otherwise.

II. Advantages of OPC-UA

The fourth industrial revolution is about making machines and products "smart" to make decisions and control the manufacturing process on their own [1]. To make these decisions they need to be connected to other machines and resources to get data on which the decisions are based. On this need for connection many different standards where build. Two of the most common are Message Queuing Telemetry Transport (MQTT) and OPC-UA [2]. Both are build on the Transmission Control Protocol (TCP). UA is a standard published by the Open Platform Communication Foundation in 2008 and is an "platform independent service-oriented architecture that integrates all the functionality of the individual OPC Classic specifications into one extensible framework" [3]. While MQTT is event driven OPC-UA is has both a Remote Procedure Call (RPC) and an PubSub mode [3]. OPC-UA is not as lightweight as MQTT but the information model is much more semantic and it is even much more performant than MQTT [6]. In 2014 the Fraunhofer IOSB has started the open62541 library for C in cooperation with RW-TH Aachen and the TU Dresden [5]. Today this is one of the feature richest open source implementation of the OPC-UA standard [4]. On the other hand there is FreeOpcUa as an very easy to use python library. Its performance might not be as good as the C-library but with an powerful PC it isn't the limiting factor and can also be used.
The architecture of OPC-UA with an standard between all servers and clients makes it perfect for building a decentralized and interoperable network. It is easy to have multiple servers and clients in one network and every client can talk to every server simultaneously. This makes the network very flexible.

III. Development of the data connector

Depending on the robot it might be possible to install software onto it. If this is the case there is no need for additional Hardware . On the Husky PC runs ROS, so it is possible to create an ROS package with a OPC-UA Server. On the Kinova robot arms on the other Hand it is not possible to install software, so an Raspberry Pi is used which gets the data from the robot and makes them accessible via an OPC-UA server. To simplify the development the OPC-UA Server code is implemented as a stand-alone class, so it can be reused for different robots.

A. As a ROS package

B. As a stand-alone device

As Hardware for the stand-alone device a Raspberry PI 3B+ is used because it is very versatile. It could connect to robots via Ethernet, USB or with the GPIO pin nearly every other connection standard. Furthermore it is powerful enough to handle an OPC-UA server and talk to an robot at the same time and it's small. Because of its size it can be mounted directly on the Kinova robot without interfering with the movement of the robot (Fig. 1). The data connector can even be powered by the USB ports on the Kinova robot, so no additional power source is needed.
The Code is split into three main parts (Fig. 2). One part is

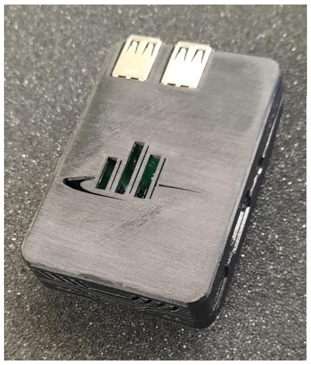

Fig. 1. The data connector mounted directly on the Kinova robot

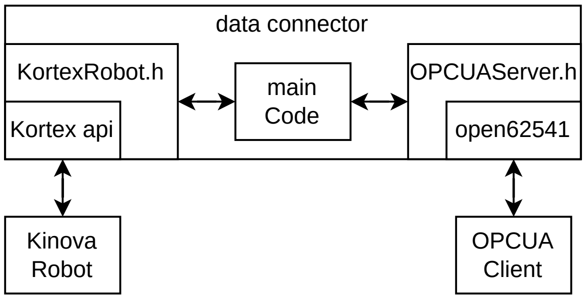

Fig. 2. Code structure of the data connector

the communication to the robot via the Kortex api, the second part is the OPC-UA server with the open62541 library. The connection to the robot is done on the UDP protocol and the data is requested via the cyclic feedback function of the Kortex api, because it is the fastest way to get the data from the robot. The main code part in the middle starts a thread for the connection to the robot and a thread for the OPC-UA Server. It also connects the data from from the robot class to the OPC-UA Server class. To do this there is a map defined that connects the function needed to get the data from to robot to the name of the variable on the OPC-UA server. Therefore it is very easy to add or remove variables from the OPC-UA Sever by adding or removing an entry from this map.
This system allows for a lot of flexibility. The data connector is plug and play with any of the Kinova robot arms because they all use the Kortex api. If the connector is to be used with another robot from another manufacturer, it is enough to just write a new connection to the robot and modify the main code. It is also possible to prepare the connection protocol for multiple robots from different manufacturers and let the main code detect which robot is connected and which protocol to use.

IV. Performance Tests

To ensure the data doesn't take to long from the robot to a client a series of performance tests are conducted. These can be separate in two parts. First getting the data from the robot to the data connector and second get the data from the data connector to the client. Every test is made with 5 samples. Every sample is the average speed of the first 1000 data requests.

A. Getting data from the robot

The Raspberry Pi 3B+ is directly connected to the Kinova robot via a ethernet cable. With UDP as communication standard the Raspberry Pi can get the up to 1400 datasets per second from the Kinova robot (Fig. 3). That is even more than the 1000 datasets per second, that Kinova claims [7]. A dataset consists of all the data the robot has to offer. Thats over 50 data points. Is the OPC-UA server is running parallel on the Raspberry Pi the frequency is lower, around 1kHz. If one client is simultaneously requesting data from the OPC-UA server the frequency isn't really affected. But if there are more clients then the frequency begins to drop slowly.

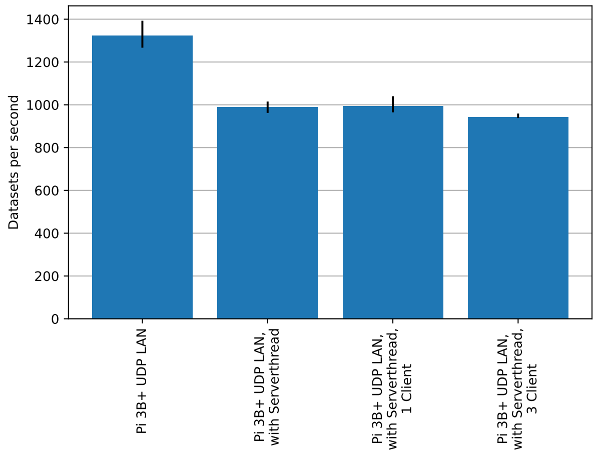

Fig. 3. Frequency with which the Raspberry Pi 3B+ can get data from the Kinova arm

B. Getting Data from the data connector

To get the data from the data connector two different PCs are used. One midrange Laptop (Intel 11.Gen Core i7-1165G7, Realtek Semiconductor RTL8152 Fast Ethernet Adapter) and an high end Tower PC (Intel 13.Gen Core i9-13900KF, Realtek Gaming R2.5GbE Family Controller). The PCs and the data connector are all connected to a Router (tp-link Archer C80 AC1900 MU-MiMO).
In Fig. 4 different libraries are used for the client to request one Value from the data connector. On the Laptop the c-library open62541 is faster than the python library freeopcua. On the Tower PC the python library is even faster than the c library on the Laptop. If these results are compared to the ping between these PCs and the data connector (Fig. 5) it can be concluded that the python library on the Tower PC is running close to the actual limit of the network.
If multiple clients are started in parallel on the Tower PC there is a slight decrease in the frequency with which the data can be requested from the data connector (Fig. 4). In the last test (Fig. 6) datasets with different amounts of values are requested from the data connector via the Tower PC using the python library. The difference between 1 Value an 10 Values in one request

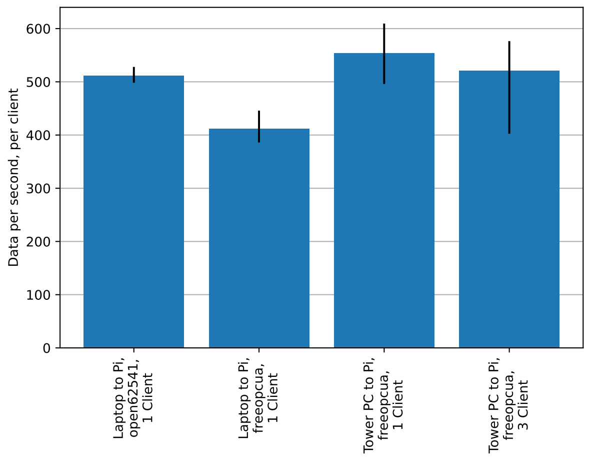

Fig. 4. Frequency with which data can be requested from the OPC-UA server

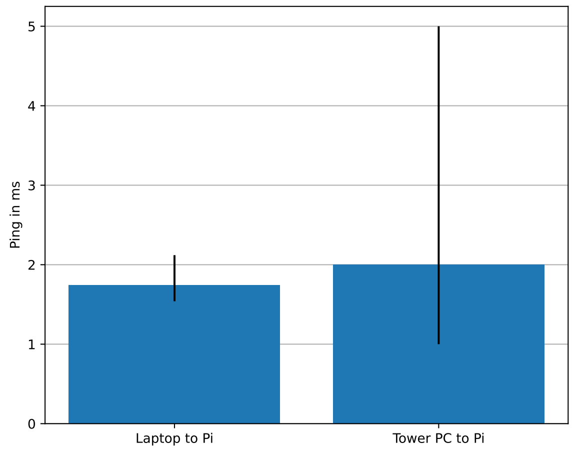

Fig. 5. Ping to the data connector

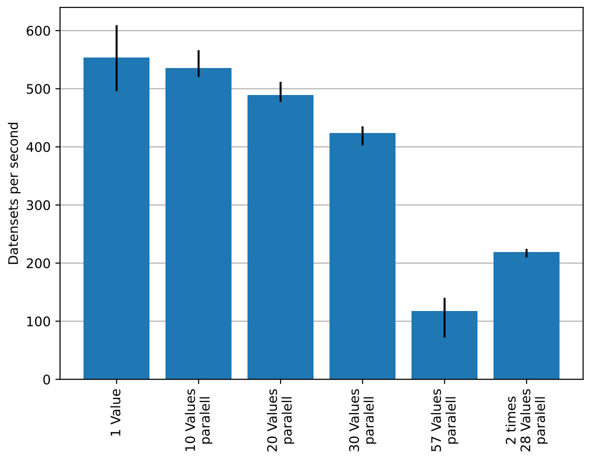

Fig. 6. Frequency with which different datasets can be requested from the OPC-UA server

is relative small (18 packages), between 10 and 20 Values this gap widens (47 packages) and between 20 and 30 it is even bigger (64 packages). It is clearly not an linear decrease but also not an exponential one. Both the PC and the Raspberry Pi are using less than 20% CPU so this isn't the bottleneck. The size of the package gets bigger with every additional Value and there seems to be no limit at which the packages are split. If all 57 Values are requested in one package its size is 2538 bytes. This length is depending mostly of the name of the Values because the longer they are, the bigger the package.
The best way to request this many Values is in two packages with 28 and 29 Values each. The frequency is half that of one 30 Value package but this is still nearly two times as fast as requesting all 57 Values in one package.

V. Husky battery SoC

The State of Charge (SoC) display of the Husky is probably tuned for an lithium ion battery so it is very inaccurate for the sealed lead-acid battery that is currently powering the Husky. Because of this tuning it shows only around 50% SoC with an full battery and after a short usage it shows 0% SoC. To create a better estimation of the SoC a ROS package is created that uses the measured current an potential of the battery and publishes an estimation of the SoC tuned for a SLA battery. This data is also published on the OPC-UA server.

A. *Estimating the SoC*

In the documentation of the Husky is Fig. 7 to be found. With this curves the SoC can be estimated using the measured

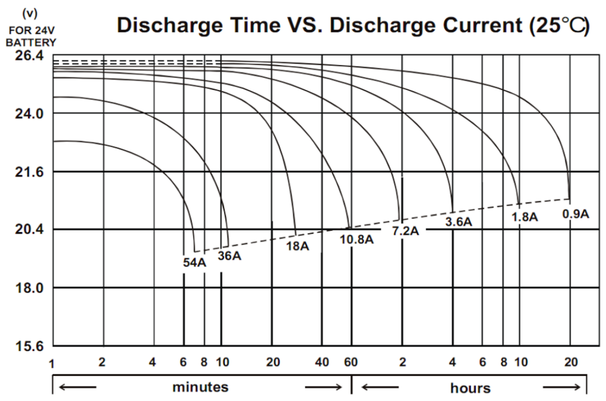

Fig. 7. Discharge time over different currents for the SLA battery of the Husky [8, p.21]

current and potential of the battery. First the functions of the curves have to be approximated. For this a 4th degree polynomial is used. It represents the curves fairly accurate within the range of the curves. To make the estimation easier the x and y data is flipped before approximation. The results are functions for different current that output the time the battery is used with this current. Comparing this value with the maximal time a the battery can be used with this current gives the percentage of battery that is used.

B. *Validating the SoC estimation*

To validate the estimation a battery is put thur an complete discharge cycle and the estimated SoC, the current and the potential of the battery are recorded. In Fig. 8 is the estimation of the SoC over one discharge cycle shown. As a comparison

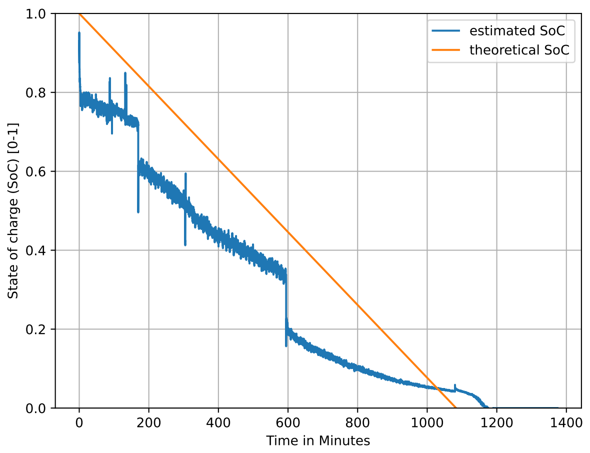

Fig. 8. Comparison between the theoretical SoC and the estimated SoC over one discharge cycle

there is also the theoretical SoC calculated from the average current shown. First of all it is visible that the battery doesn't start at 100% it is likely that the battery doesn't hold a full charge anymore because of wear. The estimated SoC is an moving average over 100 values but still there is some noise in the date. Thats because the measured potential and current are very noisy (Fig. 9). There are also big spikes in the data

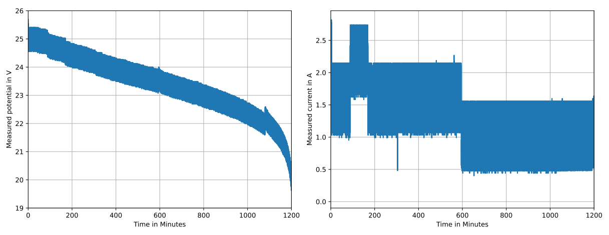

Fig. 9. Measured potential and current over one discharge cycle of the Husky battery

around 180 and 600 minutes. At the same time there are spikes in the current but not in the potential. For this reason it is likely that this is based on measuring error. Because there is no documentation on which sensor Clearpath has used it can't be replicated. But the the new SoC estimation is much better than the old one and an underestimation of the SoC leads to an bigger safety margin which is also good.

VI. Digital twin

The digital twin is build in Isaac Sim. This is a simulation tool for robots which part of the NIVIDIA Omniverse. In order to connecting to the OPC-UA server, a client is needed. The best way to implement this is via an extension for Isaac Sim. To reflect the modularity of robot system and the decentralized Network approach multiple different configurations of the robot system are created. The extension has an Graphical User Interface (GUI) in which the configuration of the robot system can be chosen. Independent form the chosen configuration it is also possible to choose to which OPC-UA server you want to connect. This allows to test a robot configuration you don't have. For example it could be tested if the Husky will tip over when the Kinova arm does a specific movement, even if you don't have an Husky robot. Just load an configuration with the Husky and an Kinova arm and connect to the Kinova arm. The Kinova arm can now be moved with the controller and you can test any movement.
Furthermore it is possible to open a graph in which any data from the robot can be displayed. So even the non-visible data can be visualized. Even if you don't use one of the robots in the robot system it is still possible to connect to the robot to keep an eye on its vitals.
To make the digital twin more realistic you can choose in the GUI to load a 3D scan of the University as an environment for the digital twin (Fig. 10).

Fig. 10. Comparison of the digital twin (left) and the real robot (right)

VII. Remote access

To monitor the robot system from everywhere in the world a remote access to the data connectors is needed. For this an mobile 5G router is mounted on the Husky. To this Router all the data connectors are connected. On the router an (name?) VPN is installed, so to connect to a data connector a client has to connect to the router via the VPN first. Afterwards the connection to the data connector is identical as before. In Fig. 11 an Example is shown where the client is accessing the battery data of the Husky from a train while the Husky is in the lab of the university. To achieve this a special Sim Card

Fig. 11. Example of the remote access to the data connector of the robot in the university lab from a Train

for the router is needed. It has to have an public IP-Address. For Sim Cards this is not common. Normally they get their Messages like push notification via a Google or Apple Server. Without an public IP-Address it is not possible to reach the VPN server.

VIII. Conclusion

It could be shown in this paper that it is possible to create an decentralized and interoperable network using the OPC-UA standard. Furthermore with the data connector the whole setup is very flexible because it can be added to every Kinova robot arm using the Kortex api. It is also possible to expand the data connector to work with many different robots from different manufactures. The shown digital twin proves it is easy to connect to different robots when they are all using the same standard and the performance tests have shown this standard is fast enough to deliver a accurate digital twin. Thur the possibility of an remote connection to the different data connector is with this setup possible to monitor the robot system from all over the world.

References

[1] Lasi, H., Fettke, P., Kemper, HG. et al. Industry 4.0. Bus Inf Syst Eng 6, 239–242 (2014). https://doi.org/10.1007/s12599-014-0334-4

[2] P. Marcon et al., ”Communication technology for industry 4.0,” 2017 Progress In Electromagnetics Research Symposium - Spring (PIERS), St. Petersburg, Russia, 2017, pp. 1694-1697, doi: 10.1109/PIERS.2017.8262021.

[3] Open Platform Communication Foundation, Unified Architecture, https://opcfoundation.org/about/opc-technologies/opc-ua/, last checked 09.02.2024.

[4] Mühlbauer, Nikolas; Kirdan, Erkin; Pahl, Marc-Oliver; Waedt, Karl (2021): Feature-based Comparison of Open Source OPC-UA Implementations. INFORMATIK 2020. DOI: 10.18420/inf2020_34. Gesellschaft für Informatik, Bonn. PISSN: 1617-5468. ISBN: 978-3-88579-701-2. pp. 367-377. 5th GI/ACM I4.0 Standardization Workshop on Industrial Automation and Control Systems. Karlsruhe. 28. September - 2. Oktober 2020

[5] Florian Palm, Sten Gruner, Julius Pfrommer, Markus Graube, Leon Urbas, open62541 – der offene OPC UA Stack, https://publica-rest.fraunhofer.de/server/api/core/bitstreams/d860fd4c-c051-4e71-815b-23c7fc4c2549/content

[6] Profanter, Stefan, Tekat, Ayhun, Dorofeev, Kirill, Rickert, Markus, Knoll, Alois. (2019). OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols. 10.1109/ICIT.2019.8755050.

[7] Kinova Robotics Inc., Kortex TransportClient classes, https://docs.kinovarobotics.com/kortex/linked_md/cpp_transport_router_session_notif.html#transportclient-classes, last checked 08.02.2024.

[8] Clearpath. Husky A200 UGV User Manual. url: https://clearpathrobotics.com/husky-documentation/ last checked am 21. 10. 2023.