Optimizing Camera Stream Transport in Cloud-Based Industrial Robotic Systems

Marcell Balogh and Attila Vidács

Abstract—**Combining visual-guided robotics with cloud networking brought a new era into industrial robotic research and development. New challenges have to be tackled with a focus on providing proper communication and data processing setup: sensor data processing as well as the control software should be decoupled from the local robot hardware and should move into the cloud. In the emerging field of cloud robotics, there are trade-offs that have to be handled. More and more sensors such as cameras are being integrated but it comes with a cost. All sensory data have to be sent through often limited networking resources, while latency must be kept as low as possible.**

In this paper we propose a general solution for efficient camera stream transportation in cloud robotic systems. After introducing our test scenario with the used hardware and software elements, a detailed overview of the architecture is presented with describing each task of the components. The goal of this paper is to examine the current stream transportation implementations in ROS environment and implement a more efficient method. The performance of the proposed method is investigated and compared with other solutions evidenced by measurements.

Index Terms—**cloud robotics; distributed systems; image processing;**

I. Introduction

CONTRARY to certain expectations that human workers will be displaced by robots, the real trend is to utilize collaborative robots beside humans to work with. This change necessitates robots to be aware of their full surroundings real time, that seems to be the real challenge.

Vision systems are widely used in industrial robotics for various tasks and processes such as inspection and quality control, robot guidance, safety of workers, assembly lines, etc. As a result of the continuous improvement of camera sensors, the size of the raw sensory information significantly increased. It became a trade-off between the camera quality and the latency of the transported image stream. In order to get the best performance, modifications need to be tailor-made.

We expect these systems to examine their environment through various sensors and act immediately to prevent human injuries or collisions. Having applied the techniques of cloud computing, image processing, robotics and distributed networks, we present an alternative stream transportation method for vision aided real-time robotic systems. Our approach is to combine the existing methods in a more efficient way with distributed systems.

Throughout this work we utilized the Robot Operating System (ROS) [1] which became the *de facto* standard of robotic development. ROS based systems fit well into the concepts of cloud architecture. Sensors are providing real-time information which are being sent over the network, while data are processed in the cloud, and only the acting commands are being sent back to the robot. In this scheme, complex sensor networks can be easily managed, and data could be processed off board.

The paper is outlined as follows. In Section II, the background of the applied technologies is introduced. Section III presents an overview of the system including the hardware devices and the software design. Section IV explains the demonstration task, the steps of image processing alongside with the network setup. In Section V the different measurement methods are presented with their arrangements. This is followed by the performance measurement in Section VI which contains a detailed comparison of the different options. It contains the overall result and performance of the implemented system. Finally, conclusions and considerations regarding the improved solutions are presented in Section VII.

II. Related Work

Industrial robots took a long way to reach their current form and the new era of collaborative robots is currently rising. On the contrary of what people believed with the appearance of industrial robots, workers are still essential elements in the factories thus a stronger human-robot cooperation has started to emerge. Robots are able to cooperate even better with humans, taking their presence into account and proceed with caution.

To help robots look around and act within their environment, visual servoing is a popular approach. Visual servoing is an approved technology which was first proposed in 1996 by Hutchinson *et al*. [2], and was significantly improved by F. Chaumette and S. Hutchinson [3]. Nowadays two popular approaches were formalized: Position Based Visual Servoing (PBVS) and Image Based Visual Servoing (IBVS). PVBS seeks to calculate and minimize errors in the global reference frame while IBVS minimizes errors in the image plane of the camera.

Autonomous navigation for mobile robots is a prevailing topic among robotic researchers. To tackle with challenges, Kalman Filter-based solutions are the leading methods for sensor fusion [4]. Nguyen *et al*. proposed a solution for

Marcell Balogh is with the High Speed Networks Laboratory at the Department of Telecommunications and Media Informatics, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Hungary, (e-mail: balogh.marcell@edu.bme.hu)

Attila Vidács is with the High Speed Networks Laboratory at the Department of Telecommunications and Media Informatics, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Hungary, (e-mail: vidacs.attila@vik.bme.hu)

DOI: 10.36244/ICJ.2022.1.5

autonomous navigation based on Robot Operating System and Gazebo [5]. They used deep learning with simulation data to improve real-world applications. Authors in [6] proposed an improved algorithm with Extended Kalman Filter (EKF) to estimate the state of an Unmanned Aerial Vehicle (UAV) in real time.

To be able to select the optimal streaming method, a comparison was carried out in [7]. After thoroughly comparing H.264 Advanced Video Codec, Dirac, Theora and Motion JPEG2000, the study clearly indicated the advantage of inter-frame comparison in H.264.

Visual aided robots require a well-considered approach for a proper real time stream forwarding. Video streaming over cellular or wireless networks also appears in various fields. A popular and versatile tool for stream forwarding is to utilize GStreamer [8]. A typical use-case of GStreamer is real-time video streaming but it is also useful for acoustic signal processing [9] or even for detecting gravitational waves [10].

Examining the capabilities of a closed-loop control over imperfect networks were investigated by Rácz et al. [11] They worked with an Universal Robots UR5 manipulator to test and measure the performance impact of Ultra Reliable Low Latency Communication (URLLC) capability in 5G networks. As a result, measurements showed that network delay lower that 4 ms has no significant performance impact in case of the robot arm.

III. System overview

The selected use case presents a visual guided robot arm manipulation task, where the emphasis is on the near real time control of the manipulator, based on visual information. However, a large variety of different applications can be realized using the same design patterns that the ROS ecosystem provides.

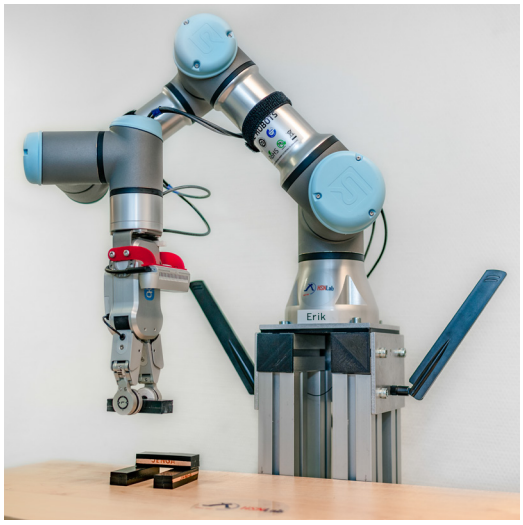


Fig. 1. Realized robotic system: An UR3e industrial robotic arm equipped with a RealSense D435i depth camera.

Our test scenario consists of an UR3e collaborative robotic arm from Universal Robots [12], an Intel RealSense D435i depth camera [13] and Raspberry Pi4B devices to host the camera driver. As an end-effector for the robotic arm, an OnRobot RG2-FT gripper [14] is applied for pick and place tasks. As Fig. 1 shows, the depth camera is rigidly mounted onto the last joint of the robot arm following the eye-in-hand approach to be able to inspect both the surroundings and the gripped object.

The software implementation is based on the Robot Operating System and follows the cloud robotic aspects as Fig. 2 presents. It provides a framework with the most common communication patterns for a distributed system like publish-subscribe or request-response. The ROS structure consists of a ROS Master running in the cloud, an image forwarding node on a local hardware element (RPi4), and an image processing node placed in the cloud.

Generally, ROS offers raw image transport but it can be extended with plugins to support JPEG or PNG compression. In case of the RealSense camera, an official ROS package [15] is available for retrieving data through topics to work with. Besides ROS, there is an open-source solution for depth cameras to be networked over wired Ethernet or Wi-Fi connection [16]. To help object detection, OpenCV library[17] is used both for colour and depth image processing.

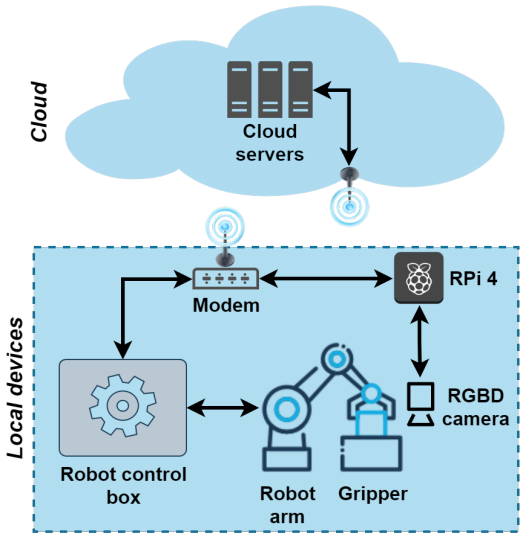


Fig. 2. System architecture.

IV. System realization

In order to demonstrate the system capabilities, our robot is programmed to build a tower from small identical wooden blocks in a jenga tower style. The task for the robot is to automatically detect jenga pieces on the table, then grab and place them on top of the already built structure.

The detection task includes two different phases. First, when a new jenga block is detected by the camera, its position is acquired in order to be able to pick it up. To achieve this, only the colour frame is used with filtering for colour ranges. The other part that requires the camera is when the robot wants to place the jenga onto the tower being built. It examines the

target area whether a previous element or elements have been already placed using the depth camera image. Being conscious of the 3D coordinates of the top tower elements will help the actual block to be placed in the right position.

Note here, that neither the pick position nor the follow up tower position is pre-programmed. It is up to the robot to automatically detect and decide, even making it possible for a human to join in and build the jenga tower together with the robot in a cooperative manner. The robot will put its own actual jenga piece on top of the structure, no matter whether the user just placed a new piece atop, or even removed the last one.

A. Image processing

Image processing has to provide reliable actual information to achieve the introduced demonstration task. It requires techniques handling both colour and depth information. The current problem could be separated into two phases: working with colour images and working with depth images.

1) Colour image: The task related to the colour image is to find a black jenga block on the table, and if there are more, choose one considering whether the robot can reach it or not. It is based on a robust colour segmentation process in HSV colour space. The black colour of the jenga block and its non-reflecting surface makes accurate filtering for black colour possible. It also excludes objects where there is a measurable difference of the length of the sides and the area of the object in sight. To prove its robustness, it has been successfully tested with strong light sources and with different lighting conditions. With OpenCV, the orientation and exact position have been extracted from the detection image. These are then sent to the robot to grip the object at the right pose.

2) Depth image: From the depth image, the goal is to extract the accurate distance and provide an illustrative image about the tower level measurement. Intel gives examples for depth images processing [18]. Calculating the Z distance from the depth image is helped by a function that returns the depth distance from a 2D pixel coordinate. First, an edge-preserving spatial filter is applied, which smooths the depth noise while attempting to preserve edges. In a noisy measurement it will smooth the data but could result in unwanted artifacts such as rounded or elongated edges. It is followed by temporal filtering and hole filling. It is also advised to do whenever possible, making sure not to let holes—where the depth equals zero—influence calculations. It is done by an exponential moving average (EMA) filter which is also used for spatial filtering. With an accurate parameter, we tried to reduce temporal smoothing near edges and also exclude holes.

Fig. 3 presents the before and after phases of jenga depth detection. It clearly shows that the holes were eliminated, and the contours became more accurate.

3) Equalization: Because the gripper was too close to the camera, it distracted the depth measurement. Thus, the task was to equalize the histogram of the depth image, because a 1 cm difference in a 16 bit image cannot be easily distinguished. To improve the detection, histogram equalization was only applied to a region of interest, where jenga block could occur. In Fig. 4 the distances of the tower corners are presented. Due to a shallow depth, histogram equalization results in displaying small differences with significantly changing colours. The distances are calculated as the median value of the intersecting jenga block marked with the white dots.

Fig. 3. Depth images of two jenga pieces as seen from above, before (left) and after (right) image processing.

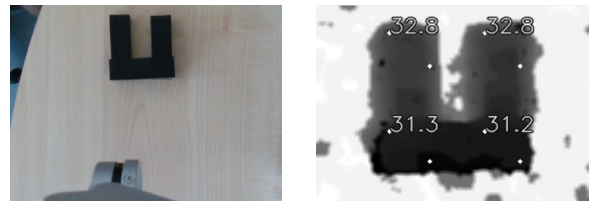


Fig. 4. Depth histogram equalization: raw camera image (left) and equalized depth image (right).

B. Network setup

All the components of the robotic system are prepared to connect and communicate using a cloud architecture with wireless radio access (such as 5G). Fig. 5 depicts the distributed software architecture of the system.

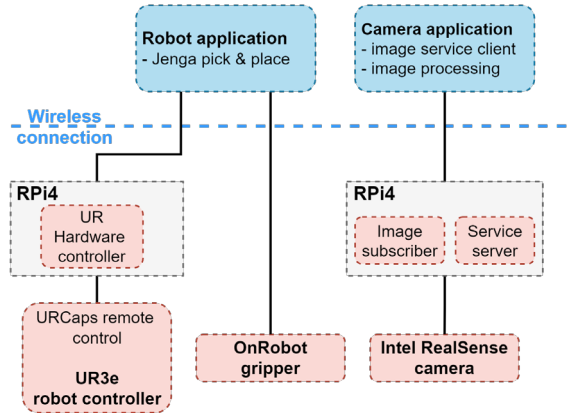


Fig. 5. Distributed software architecture with cloud elements.

1) Robot arm with cloud control: Although the UR3e controller can be connected directly to the network, the challenging part is to perform the low-level (servo-)control the robot from the (edge-)cloud. Due to its ultra-low latency requirement, it is not yet possible to run the low-level control from the cloud. (Later, with the improvement of the experi-

mental 5G network parameters, it could also be moved to the cloud.)

2) Camera and cloud image processing: A Raspberry Pi device (RPi4) serves as a gateway and driver host for the RealSense camera, since the camera hardware lacks any direct network connectivity.

Note here, that although the publish-subscribe scheme publishes every camera frame, the image processing takes only one frame from the stream to work on. hence, there is no need to to transfer the camera stream to the cloud continuously. Taking this into account, live streaming can be reduced to give a frame only when it is required. Because of the camera has a warm-up time, the idea is to combine the publish-subscribe communication scheme with a service-based request-response phase to fetch the image frame as Fig. 6 presents.

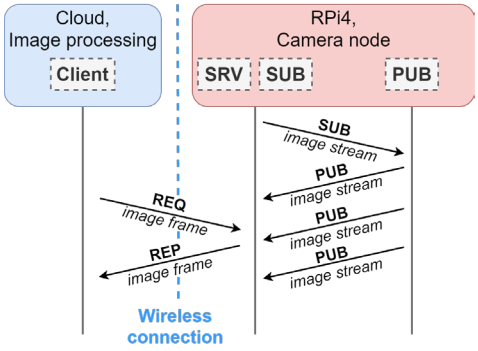


Fig. 6. Combined publish-subscribe and request-response (aka. client-server) communication for acquiring an image from the camera for cloud processing.

According to this enhancement, the camera driver node running in the background always provides the camera stream, but it will not transfer the stream continuously over the network but only forwards it to a local service server. This server provides only one image by grabbing a frame from the published camera stream in case of an incoming request from the cloud. This frame is then processed in the cloud which serve as a basis of further robot movement calculations.

V. STREAMING AND IMAGE TRANSPORT SOLUTIONS

Here we propose a custom method for image transportation that offers low bandwidth and latency. A hybrid method is also introduced with the combination of different network schemes. Alongside with our implementation, popular image transportation methods are introduced that are common in robotic applications.

A. ROS streaming

To transmit data among nodes, ROS by default use serialization and sends raw image objects as string which is a less effective way of image transportation and streaming. As measurements show, it takes 25 MB/s to transfer raw colour and depth images with 30 FPS. ROS plugins offer compression methods for a more effective image transport. It can be configured for PNG or JPEG compression. The compression range in case of the JPEG is [1, 100] where lower values trade image quality for bandwidth savings. To find the appropriate balance between image size and quality, parameters needs to be fine-tuned according to the application's needs.

B. Streaming via GStreamer

GStreamer is a current open-source multi-platform multimedia framework with widespread API options like the OpenCV library, OpenGL, RealSense or other community-driven projects [19]. It became popular for streaming audiovisual content and has been frequently used in studies relating to video transmission for its utility and flexibility in the delivery of audio and visual content. The multimedia pipelines of GStreamer are completed through the use of plugins which are assembled using so called pads to form an interconnected framework, moving video from the sink pad of a plugin to act as the source of another.

The common part in each image processing solution is the utilization of OpenCV. To exploit its capabilities, OpenCV is compiled on both sender and receiver side with GStreamer integration. The goal of GStreamer in this project is to separate the application (e.g., video player, video editor) from the streaming media complexity (e.g., hardware acceleration, remoteness). In this case only the video transmission is relevant without the audio track.

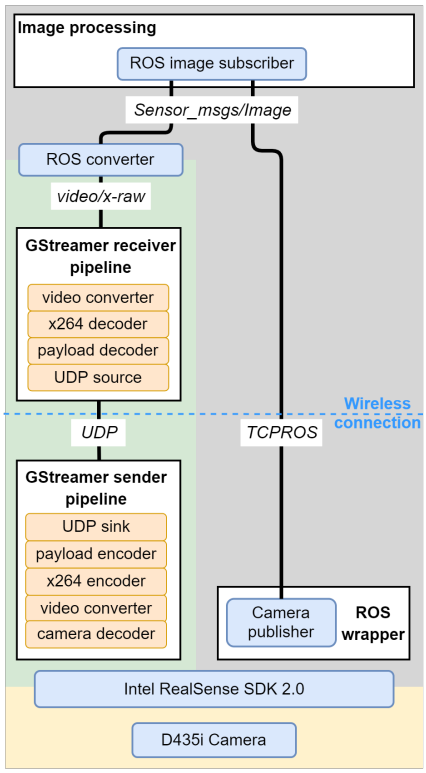


Fig. 7. Streaming and image transport solutions software architecture.

Regarding the restrictions and specialities, the streamer pipeline looks as Fig. 7 presents. Because of the pipeline forwards preprocessed images from OpenCV, the frames inserted directly from script to the GStreamer pipeline. To

encode camera stream, H.264 encoding is applied. Instead of encoding frames individually, H.264 compress across frames. This inter-frame compression significantly reduces bandwidth consumption because most frames record only the changes from the previous frame. In order to be able to send the H.264 stream over the network, it needs to be encapsulated into Real-time Transport Protocol (RTP) packets [20]. After providing the H.264 payload, it is combined with UDP, and is ready to be sent through the network.

In receiver side the source is an UDP connection with a related port number. Upon receiving the datagram, the decoder extracts H.264 video from RTP packets and sends it to another converter which creates a raw formatted stream with the requested parameters (640x480 resolution, 8-bit grayscale) which could be managed by OpenCV. After the stream is handed over to OpenCV, it is ready for ROS integration. With OpenCV, the image stream is then packed into a ROS compliant *sensor_msgs/Image* type message. After this conversion, the image stream is being published on a ROS topic.

C. Streaming via EtherSense

Intel provides an open-source solution for its D435i depth camera for connecting to the network by using a Raspberry Pi device [21]. It applies industry standard RTP protocol for streaming. Raspberry Pi only responsible for the transportation part, meanwhile post-processing such as depth decimation or spatial and temporal filtering could remain in the cloud.

According to the measurements, the bandwidth for both colour and depth stream with 640x360 resolution and 30 FPS requires 27.6 MB/s. By exploiting the full potential of the camera, with 848x480 resolution and 90 FPS, the required bandwidth reaches 146 MB/s. This option can easily saturate the available networking bandwidth, thus it does not provide a scalable solution for mass-robotic application of visual guidance.

VI. Measurements and Performance Evaluation

Next, we present performance measurement results of our system with the different camera image transport solutions. Besides end-to-end latency as being perhaps the most important transport performance parameter, the required transmission bandwidth is also measured and compared with different network and image parameters, then the advantages and limitations are evaluated.

A. Required bandwidth

An important performance indicator of the realized system is the required bandwidth to operate with. First, the ROS streaming solution is measured, then it is compared with our GStreamer solution to present the results.

For traffic measurement purposes, we used the internal topic bandwidth monitor on ROS side, while for Gstreamer traffic the real time Linux network bandwidth tool called Interface TOP (IFTOP) was used. To measure exclusively the image transport, the actual port was filtered where GStreamer is forwarding the stream. All the measurements reflect an average of 10-minute data traffic.

The performance of ROS streaming is summarized in Fig. 8 where all the measurements were done with 640x480 resolution and 30 FPS. As expected, the raw image transport over ROS has the highest bandwidth requirement. These results are close to Intel's official EtherSense realization, where for example the depth streaming with similar parameters took 13.88 MB/s.

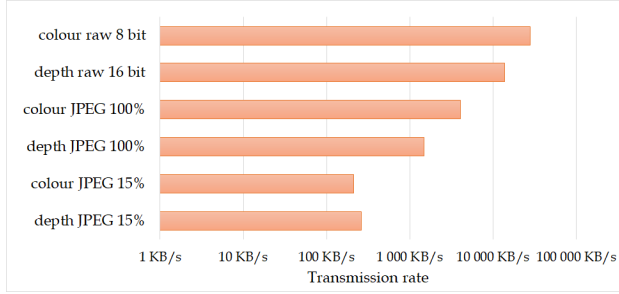


Fig. 8. ROS image streaming comparison.

Although compressed image transport could significantly lower the data rate, the hybrid method could present an even better solution. The idea of providing only one frame on request could minimize the data transfer. However, the image streaming is still running locally in the background, but does not flow into the cloud unless it is requested.

The JPEG compression methods can also be applied to the requested frame as well. This results in such a small image size that could be easily transferred without any difficulties. The size of only one colour frame with 15% JPEG quality is approximately 215 kB, the corresponding depth frame size is 265 kB.

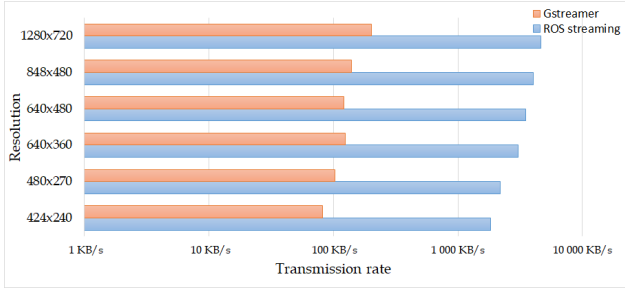


Fig. 9. Depth image streaming comparison.

Comparing these results with the ROS streaming options, GStreamer could perform better up to more than an order of magnitude. Rather than compressing images one by one as ROS does, GStreamer uses H.264 encoding where the inter-frame compression significantly reduces the bandwidth consumption. (Note, that static scenes with mostly still frames encoded with H.264 can result in very low data volumes, therefore the measurements used contain both still scenes and scenes in motion to collect more meaningful data.) GStreamer was set up to forward with constant quality. To show a more revealing chart and visualize the differences, logarithmic

scaling is applied in the figures.

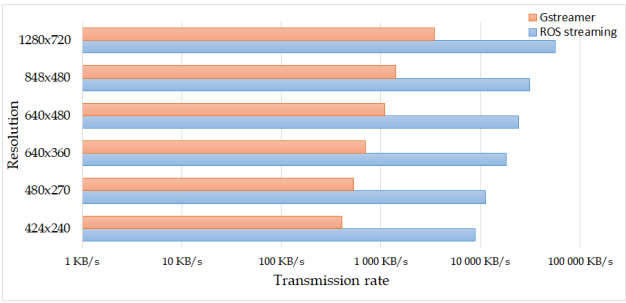


Fig. 10. Colour image streaming comparison.

While examining the streaming of colour images, the parameters and measurement method remained unchanged. Here the results (see Fig. 10) show the overhead of ROS streaming against GStreamer.

Taking into account the image quality and the required bandwidth, it is clearly a trade-off. According to our experiences, among all the available ROS image transport methods, the 15% JPEG option with 640x480 resolution turned out to be the best trade-off between image quality and bandwidth usage, when detecting a jenga block. Thus, the following measurements were all carried out in 640x480 resolution.

B. End-to-end latency

For latency measurements, glass-to-glass method was applied: it measures the time it takes between the moment of action happens in front of a camera (first glass), and the moment that a viewer sees the result of this action on the screen (second glass). To determine latency, an online stopwatch were used meanwhile the camera was faced to the notebook screen and forwarded the encoded stream for cloud processing (see Fig. 11). This method makes end-to-end latency measurement possible, including such elements like the camera readout time, encoding and so on. On the other hand, with this method a possible bottleneck is the 60 Hz refreshing rate of the screen. Because of the camera relies on the content of the screen, it could not be more accurate than 1/60 s (approx. 17 msec).

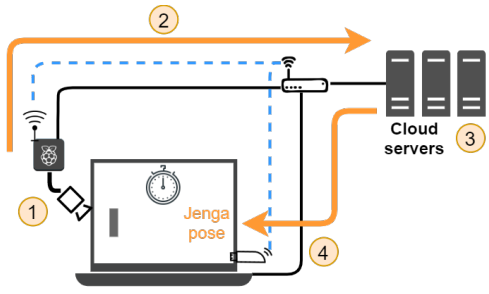


Fig. 11. Latency measurement setup and methodology (wireless connection: blue dotted line, wired connection: straight line).

During the measurement, the camera detects an appearing jenga (see step 1 on Fig. 11) from a pre-recorded video shown on the laptop screen, which also displays a running stopwatch. The camera stream is then forwarded into the cloud (step 2) where the image processing takes place (step 3). Finally, the result is sent back and displayed on the laptop screen (step 4).

The critical component of an end-to-end latency is the radio transport delay over the wireless link. In order to be able to examine this component, reference measurements were taken where all wireless links were replaced by wired connections (see Fig. 11). Table I presents results collected from 10 different measurements for both the wireless and wired options where the average latency and its standard deviation are listed for the two image transport methods.

TABLE I
END-TO-END LATENCY MEASUREMENT RESULTS

	ROS transport		GStreamer	
	wireless	wired	wireless	wired
average	2286.9 ms	1123.7 ms	244 ms	222.3 ms
standard deviation	383.1 ms	120.2 ms	66.2 ms	32.3 ms

The highest average delay is measured for ROS transport over wireless. Its value is two times higher that the experienced wired transmission delay. The standard deviation for the wireless ROS transport is also three times as high as for its wired counterpart. However, the end-to-end delay for the GStreamer solution does not show significant difference for the two cases, although the standard deviation is doubled for the wireless case. Note here, that the standard deviation of end-to-end latency can be directly translated to delay jitter, which is an important performance metric for video streaming. The most important finding here is that the average delay for the ROS transport is significantly higher than the GStreamer option.

VII. CONCLUSION

In this paper, the standard ROS-based network image transmission method was examined, and an efficient custom image transport solution based on GStreamer was proposed and evaluated through a demonstration use case. The trade-off between image quality and network transmission bandwidth was highlighted. Comparing our streaming implementation with the industry-standard ROS solution, our method showed better performance by one order of magnitude. Even in wireless systems our method showed significantly lower latency. When continuous image transfer is not required, the bandwidth usage can be even more reduced when—instead of forwarding the whole stream over the network—only a single frame is sent for image processing.

Visual-aided robotics is a rapidly emerging field in Industry 4.0 solutions, thus efficient video streaming and image transmission techniques together with cloud-based image processing will form the backbone of such applications.

ACKNOWLEDGMENT

This work has been supported by the Ericsson-BME 5G Joint Research and Cooperation Project, partly funded by the National Research, Development and Innovation Office, Hungary, project number 2018-1.3.1-VKE-2018-00005.

REFERENCES

[1] Stanford Artificial Intelligence Laboratory et al. Robotic operating system. 05 2018. URL https://www.ros.org.

[2] S. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):651–670, 1996. DOI: 10.1109/70.538972.

[3] Chaumette and Seth Hutchinson. Visual servo control, part i: Basic approaches. *IEEE Robotics and Automation Magazine*, 13:82–90, 2006. DOI: 10.1109/MRA.2006.250573.

[4] Dobrev Yassen, Flores Sergio, and Vossiek Martin. Multi-modal sensor fusion for indoor mobile robot pose estimation. *Position, Location and Navigation Symposium (PLANS)*, pages 553–556, 04 2016. DOI: 10.1109/PLANS.2016.7479745.

[5] Anh Nguyen, Ngoc Nguyen, Kim Tran, Erman Tjiputra, and Quang D. Tran. Autonomous navigation in complex environments with deep multimodal fusion network. *International Conference on Intelligent Robots and Systems (IROS)*, 07 2020. DOI: 10.1109/IROS45743.2020.9341494.

[6] Hao Du, Wei Wang, Chaowen Xu, Ran Xiao, and Changyin Sun. Real-time onboard 3d state estimation of an unmanned aerial vehicle in multi-environments using multi-sensor data fusion. *Sensors*, 20:919, 02 2020. DOI: 10.3390/s20030919.

[7] Till Halbach. Comparison of open and free video compression systems - a performance evaluation. In *Proceedings of the First International Conference on Computer Imaging Theory and Applications - Volume 1: IMAGAPP, (VISIGRAPP 2009)*, pages 74–80. IN-STICC, SciTePress, 2009. ISBN 978-989-8111-68-5. DOI: 10.5220/0001809700740080.

[8] S. Nimmi, V. Saranya, Theerthadas, and R. Gandhiraj. Real-time video streaming using gstreamer in gnu radio platform. *2014 International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE)*, pages 1–6, 2014. DOI: 10.1109/ICGCCEE.2014.6922233.

[9] Kipp Cannon, Sarah Caudill, Chiwai Chan, Bryce Cousins, Jolien Creighton, Becca Ewing, Heather Fong, Patrick Godwin, Shaun Hooper, Rachael Huxford, Ryan Magee, Duncan Meacher, Cody Messick, Soichiro Morisaki, Debnandini Mukherjee, Hiroaki Ohta, Alexander Pace, Stephen Privitera, and Madeline Wade. Gstlal: A software framework for gravitational wave discovery. *SoftwareX*, 14:100680, 06 2021. DOI: 10.1016/j.softx.2021.100680.

[10] Roald Otnes, Joachim Eastwood, and Mathieu E.G.D. Colin. Using gstreamer for acoustic signal processing in deployable sensor nodes. pages 1–6, 2015. DOI: 10.1109/OCEANS-Genova.2015.7271567.

[11] Géza Szabó, Sándor Rácz and József Petö. Performance evaluation of closed-loop industrial applications over imperfect networks. *Infocommunications Journal,* XI, 2019. DOI: 10.36244/ICJ.2019.2.4.

[12] Universal Robots UR3e. [Date accessed: 12-2021] http://design.ros2.org/articles/rosmiddlewareinterface.html.

[13] Intel RealSense d435i Specification. [Date accessed: 12-2021] https://www.intelrealsense.com/depth-camera-d435i.

[14] OnRobot RG2-FT gripper. [Date accessed: 12-2021] https://onrobot.com/en/products/rg2-ft-gripper.

[15] ROS Wrapper for Intel® RealSense™ Devices. [Date accessed: 12-2021] https://dev.intelrealsense.com/docs/ros-wrapper.

[16] Alexey Puzhevich, Sergey Dorodnicov, Anders Grunnet-Jepsen and Daniel Piro. *Open-Source Ethernet Networking for Intel® RealSense™ Depth Cameras*. 04 2020.

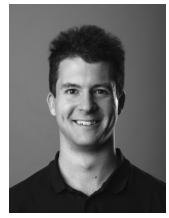
[17] OpenCV. Open source computer vision library, 2015.

[18] Dave Tong, Anders Grunnet-Jepsen. Depth postprocessing for Intel® RealSense™ D400 depth cameras. 2019. [Date accessed: 12-2021].

[19] Wim Taymans, Steve Baker, Andy Wingo, Ronald S. Bultje, Stefan Kost. Gstreamer application development manual, 2016.

[20] Wenger Stephan, Miska Hannuksela, Thomas Stockhammer, Magnus Westerlund, and D. Singer. Rtp payload format for h.264 video. 03 2005. DOI: 10.17487/RFC3984.

[21] Anders Grunnet-Jepsen, Philip Krejov. Intel® RealSense™ Depth Camera over Ethernet. 02 2019.



Marcell Balogh received his BSc degree in Electrical Engineering from the Budapest University of Technology and Economics (BME). He is specialized in cloud-based autonomous systems. In his early studies, he joined HSN Lab, a University Research Group at the Department of Telecommunications and Media Informatics, BME. Currently, he is pursuing his Master's degree on visual-aided robotic systems.

Attila Vidács received the MSc and PhD degrees from the Budapest University of Technology and Economics (BME) at the Faculty of Electrical Engineering and Informatics, in 1996 and 2000, respectively. His research interests are in the field of cloud robotics, cooperative and modular robot systems, IoT communication technologies, ad-hoc and wireless networking. Currently he is leading the Cloud Robotics Group within HSN Lab.