Proceedings of the SICE Annual Conference 2020
September 23-26, 2020, Chiang Mai, Thailand

Hardware and software video encoding comparison

Ramil Safin[1†], Emilia Garipova[1], Roman Lavrenov[1], Hongbing Li[2], Mikhail Svinin[3] and Evgeni Magid[1]

[1]Intelligent Robotic Systems Laboratory, Department of Intelligent Robotics,
Higher Institute of Information Technology and Intelligent Systems,
Kazan Federal University, Kazan, Russia
(Tel: +7-843-221-3433; E-mail: magid@it.kfu.ru)
[2]Department of Information Science and Engineering,
Ritsumeikan University, Kyoto, Japan
(E-mail: svinin@fc.ritsumei.ac.jp)
[3]Department of Instrument Science and Engineering,
Shanghai Jiao Tong University, Shanghai, China
(E-mail: lihongbing@sjtu.edu.cn)

Abstract: Long time of data encoding with software encoding mechanisms might become a significant problem when transferring digital data from cameras of mobile robots. At the same time, processor manufacturers claim that an encoding process is significantly accelerated by using a hardware encoding. This work is dedicated to a hardware and a software video encoding comparison of two state-of-the-art codecs, which were selected due to their high popularity in computer vision and robotics fields - a hardware encoding with h264_vaapi and a software encoding with FFmpeg API libx265 codec. We encoded six video sequences of different resolutions and sizes with the two codecs and evaluated obtained video quality using the Structural Similarity Index, the Peak Signal-to-Noise ratio, and the Video Multimethod Assessment Fusion metrics. Software and hardware encoding processes were also compared by CPU and memory usage, and time that was taken by the encoding process. Our results demonstrated that the hardware encoding with h264_vaapi was 5 times more memory efficient and 6 times more time-efficient relatively to the software encoding with libx265, with an insignificant difference of an output video quality.

Keywords: Computer vision, Robot vision, Video stream, Hardware encoding, Software encoding, Intel QuickSync, FFmpeg.

1. INTRODUCTION

Visual perception is one of the most important sources of external information in mobile robotics. A typical mobile robot, especially a UAV, has rather severe limitations of an available on-board power source, while broad variety of applications (e.g., Simultaneous Localization and Mapping [1], convolutional neural network based visual localization [2], path planning [3]) require intensive calculations. This issue is often approached by transferring data to a more powerful remote off-board computer, which are traditionally more powerful than on-board computers of mobile robots and thus could process collected data more efficiently [4], [5].

Generally, a remote data transfer requires encoding of original raw data, which is captured by on-board robot sensors or input devices, since the data is typically exceedingly large and often contains excessive information. Due to data redundancy, they could be compressed and encoded almost (or even) without significant information loss. An encoding is a process that translates data to some space-efficient format using various codecs [6]. Video codecs are employed in DVD players, Internet video, video on demand (VOD), digital cable and terrestrial television, videotelephony, and a variety of other applications, including mobile robots [7]. Mobile robots gather visual sensory data, encode them and transmit to remote off-board computers for further decoding and processing.

Currently, most popular video formats are H.264 [8], HEVC (High Efficiency Video Coding) [9], MPEG4, and 3GP. In our previous research [10] we developed a Real Time Streaming Protocol (RTSP) video server for Servosila Engineer crawler robot [11] using only a software encoding approach that was based on HEVC encoder and Live555 RTSP framework. Our server was written in C++. It uses Video4Linux API [12] to capture video frames from the robot on-board cameras and libx265 encoding library to compress video data before transmitting them via a network to a remote operator. A performance analysis demonstrated that on-board video encoding demands a high computation power, i.e., CPU usage. Thus, in the current study, we modified our software by using hardware resources of a computer, namely Intel QuickSync technology.

A software encoders comparison [13], which included x264 (H.264/AVC), x265 (HEVC) and libvpx (VP9) encoder implementations, demonstrated that not only x265 but libvpx could lead to significantly lower data rates over x264. These experiments demonstrated a superiority of x265 over x264 and libvpx, especially for low video resolution cases. Software and hardware HEVC encoders comparison [14] showed that the QuickSynk Video (QSV) encoder had the lowest power consump-

† Ramil Safin is the presenter of this paper.

tion. Yet, it was impossible to directly compare QSV with other encoders, as the authors used another computer for QSV tests.

A hardware-accelerated H.264 implementation could significantly reduce a generated stream bandwidth comparatively to JPEG compression [15]. According to this study, Intel QuickSync showed significant improvement in offloading a CPU, but lacked a performance improvement in dropped frames number, bit-rate, and encoding speed. If there would have been no frame drops at measured frame rates, Intel QuickSync could be stated as being superior to libjpeg-turbo. Another work on video encoding standards, which covered MPEG-2, H.263, MPEG-4, and a draft video coding standard H.264/AVC, demonstrated that H.264/AVC compliant encoders achieved essentially a same reproduction quality as encoders that are compliant with previous standards while requiring 60% or less of their bit-rate [16].

The purpose of our research is to implement video encoding using hardware and software encoders, compare them and select an effective way of encoding videos for real-time video server implementation, which could be further integrated into a mobile robot RTSP video server. The main contribution of this work was Intel processor's capabilities evaluation in reducing a CPU usage of encoding process and increasing an encoding speed while keeping a feasible video quality level.

The rest of the paper is organized as follows. Video encoders, pixel formats, and video quality metrics used in the study are described in Section 2. Software architecture and implementation details are presented in Section 3. Experimental results are evaluated, compared and discussed in Section 4. Finally, we conclude in Section 5.

2. VIDEO ENCODING

2.1. Pixel format

Pixel format describes a layout of each pixel in image data of a picture or a video. YUV420p format is based on the YUV color encoding system. The Y'UV model defines a color space in terms of one luma component (Y') and two chrominance components: blue projection U and red projection V. Y'UV420p is a planar format with Y', U, and V values being grouped together, which makes an image more "compressible": the Y'UV420p format stores image data as an array, where all Y'-values come first, followed by all U-values and finally followed by all V-values.

RGB is another popular color model of an additive type where red R, green G, and blue B primary colors' weighted combinations produce a broad variety of colors. The RGB color model is employed in sensing, representation, and displaying of images in electronic systems, such as televisions and computers, though it is also used in conventional photography.

2.2. Quality metrics

In order to compare encoders we use Peak Signal-to-noise Ratio, Structural Similarity Index, and Video Multimethod Assessment Fusion quality metrics.

Peak Signal-to-noise Ratio (PSNR) is a ratio between a maximum possible power of a signal and a power of corrupting noise that affects its representation fidelity [17, 18]. It is computed per-frame and averaged across all frames [19] and could be defined via mean squared error (MSE). Given a noise-free $m * n$ monochrome image I and its noisy approximation K, MSE is defined by Eq. 1 and PSNR if defined by Eq. 2 [20].

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \Big[I(i,j) - K(i,j) \Big]^2 \tag{1}$$

$$PSNR = 10 \cdot \log_{10} \Big(\frac{MAX_I^2}{MSE} \Big) \tag{2}$$

where MAX_I is a maximum possible pixel value, which is 255 for 8-bit pixel representation.

Structural Similarity Index (SSIM) is a method for predicting a perceived quality of digital images and videos, including digital television and cinematic pictures. SSIM measures a similarity between two images and is calculated in different windows of an image. SSIM measure between two square windows x and y of an equal size $N \times N$ is defined by Eq. 3 [21]:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{3}$$

where:
- μ_x is an average value of all pixels in window x,
- μ_y is an average value of all pixels in window y,
- σ_x^2 is a variance of all pixels in window x,
- σ_y^2 is a variance of all pixels in window y,
- σ_{xy} is a covariance of pixels in window x and y,
- $c_1 = (k_1 L)^2, c_2 = (k_2 L)^2$ are two variables that stabilize a division with a weak denominator,
- L is a dynamic range of pixel-values (typically this is $2^{\#bits\ per\ pixel} - 1$),
- coefficients $k_1 = 0.01$ and $k_2 = 0.03$.

Video Multimethod Assessment Fusion (VMAF) is an objective full-reference video quality metric developed by Netflix, The University of Southern California and The University of Texas at Austin [22]. It predicts a subjective video quality based on a reference and a distorted video sequence. The metric can be used to evaluate quality of different video codecs, encoders, encoding settings, and video data transfer. VMAF Development Kit is an open-source package on Github.

2.3. Encoders overview

A hardware encoding uses a dedicated media processor, which yields a higher performance due to a lower CPU load. The hardware is designed for a predefined set of codecs. Intel QuickSync Video uses a media processor to make a rapid video processing and conversion with a sufficient quality level. If the II-IV generation of Intel Core processors are utilized, it is possible to

use the QSV H.264 encoder. The Intel Skylake processor allows using the QSV HEVC encoder. FFmpeg provides QSV codecs and Video Acceleration API (VA-API) codecs for hardware-accelerated coding. In our research, we used VA-API implementation (h264_vaapi by FFmpeg), which allows applications to use hardware video acceleration capabilities, usually provided by a graphics processing unit (GPU). VA-API is implemented as an open-source library (libva), which is combined with a hardware-specific driver, and is usually provided together with a GPU driver. VA-API is natively supported by Intel QuickSync drivers for Linux.

The x264 codec is considered to be the leading open-source implementation for H.264/AVC encoding [13]. It is widely used by web services, television broadcasters, and Internet service providers. The x265 codec is the x264 codec adaptation for HEVC encoding, which at a lower resolution of 360p appeared to be more efficient than a software video codec library libvpx of Google and the Alliance for Open Media. In [13] authors compared x265, x264 and libvpx using PSNR-based BD-rate metrics. The comparison demonstrated that x265 had 30.8% higher results than x264 with regard to this metrics, while libvpx outperformed x264 only by 22.6%. Yet, at higher resolutions the gap between x265 and libvpx performance decreased, e.g., at 1080p they increased the performance relatively to x264 to 43.4% and 43.5% respectively (while the mutual gap decreased to only 0.1%). Considering vulnerable for data loss wireless environments, HEVC was more stable for different packet loss rates than H.264/AVC [23]. Moreover, HEVC turned out to be especially effective for low bitrates and low-delay communication scenarios.

Table 1 Comparison of acceleration technologies for a task of encoding a video source of 4 minutes length, 1080i Full HD quality and 449 MB size into 1024x768 H.264 on Core i7-2600K CPU. The table data are reported in [24]

	Encoding apps	
GPU type	CyberLink MediaEspresso	Arcsoft MediaConverter 7
Intel HD Graphics 3000	22 sec	41 sec
Nvidia GeForce GTX 570	83 sec	76 sec
AMD Radeon HD 6870	86 sec	68 sec
No acceleration	172 sec	95 sec

In [24] authors reported a test with Acrsoft MediaConverter 7 app demonstrating that the Intel Quick Sync optimized encoding significantly outperformed Nvidia's card accelerated encoding and H.264 encoding with no hardware acceleration. In experiments on Core i7-2600K CPU with 449 MB source data of 1024x768 resolution, the Intel Quick Sync accelerated encoding took 41 second with a neglectfully low CPU load, while software encoding took 95 seconds with 30% CPU load. CyberLink's MediaEspresso app with the Intel Quick Sync optimizations converted the same source into iPad playback video within 22 seconds, while for H.264 encoding with no acceleration it took 172 seconds. Table 1 structures the results of different hardware acceleration technologies comparison from [24].

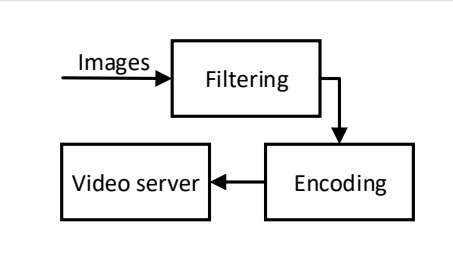

Fig. 1 Images go through filtering and encoding stages, and then are processed by the video server.

3. IMPLEMENTATION

Figure 1 presents stages of video frames' (images) encoding. These include pre-processing (filtering), encoding and transferring them to our RTSP server. The RTSP server software encoding was implemented in our previous work [10], while in this work we implement a hardware encoding.

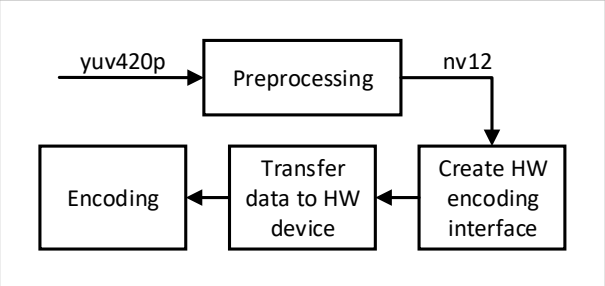

Fig. 2 The hardware encoding flow.

The hardware encoding workflow is demonstrated in Fig. 2. At a preprocessing stage raw $yuv420p$ pixel format frame is converted into $nv12$ format using FFmpeg API. Next, we create a hardware encoding interface with all encoding parameters using `av_hwdevice_ctx_create` function. An encoding codec is initialized using `avcodec_open2` FFmpeg function, and raw data are transmitted from a user space (application) to a hardware device using `av_hwframe_transfer_data` function. Preprocessed frames are issued to the h264_vaapi encoder by invoking `avcodec_send_frame` function. Finally, the encoded packets are obtained using `avcodec_receive_packet` function. In the software encoding module of our server raw input frames are converted from $yuv420p$ into $yuv420$ format at a preprocessing stage, followed by buffering the frames in order to control a frame rate, filtering (or re-sampling) in order to reduce an image size, and H.265 encoding (Fig. 3).

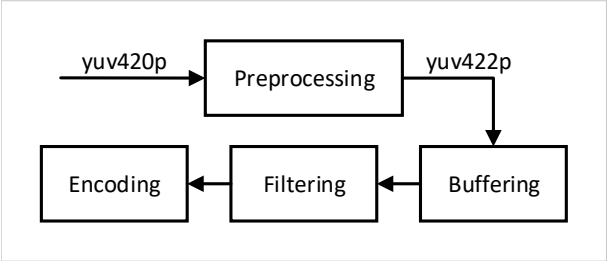

Fig. 3 The software encoding flow.

4. EXPERIMENTS

In our experiments, we encoded raw videos using the described in Section 3 software and hardware encoding approaches, and compared results using video metrics from Section 2.2. Each video was encoded 10 times with the hardware (h264_vaapi) and the software (libx265) encoders. The experimental system properties are described in Table 2. Three sample videos of 640x480 and three videos of 1280x720 resolution, all of different sizes, were selected for experiments (Table 3). All input videos had yuv420p pixel format.

Table 2 The experimental system properties

Parameter	Value
RAM	8GB
CPU	Intel core m5-6Y54
CPU frequency	2.7 GHz
Number of cores	4
OS	Ubuntu 16.04

After encoding, we compared the input and the output videos using SSIM, PSNR, and VMAF metrics. SSIM and PSNR metrics comparison was performed with FFmpeg API. VMAF was evaluated using VMAF Python library by Netflix [25]. We calculated a duration, a CPU and a RAM usage of each encoding process. The CPU and the RAM usage were computed with ATOP tool, a performance monitor for Linux [26].

Table 3 Input videos parameters.

Video ID	Resolution	Size (MB)	Bitrate (Kbps)	Duration (seconds)
1	640x480	168	92160	15
2	640x480	341	92160	30
3	640x480	728	92160	66
4	1280x720	175	276480	5
5	1280x720	445	276480	15
6	1280x720	975	276480	30

We used FFmpeg h264_vaapi encoder for hardware encoding and FFmpeg libx265 for software encoding experiments [10]. For libx265 we used an ultra-fast preset, which performs faster encoding and produces a smaller output file size comparatively to a medium preset. The output video quality of the medium preset is obviously superior to the ultra-fast preset. But since we target for real-time streaming applications (e.g., a mobile robot teleoperation task), optimizing encoding time and speed is more important than obtaining a higher quality of a resulting video. Moreover, slower presets are featured with a higher memory usage.

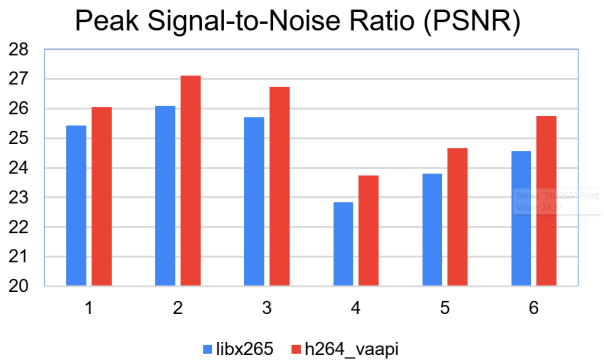

Fig. 4 Average PSNR values for six hardware (red) and software (blue) encoded video sequences. The Y-axis of PSNR is non-dimensional.

Figure 4 displays average PSNR values that were computed for the six videos. The X-axis shows a video ID, from 1 to 6. The Y-axis shows the average PSNR value (Eq. 2) corresponding to each video ID and is non-dimensional. In all cases PSNR values of hardware encoded videos were higher than of the software encoded ones. The higher PSNR value denotes a better quality.

Figure 5 displays the average SSIM values that were computed for the six videos. The X-axis shows a video ID, from 1 to 6. The Y-axis shows the average SSIM value (Eq. 3) corresponding to each video ID and is non-dimensional. In all cases SSIM values of hardware encoded videos were higher than of the software encoded ones. If the SSIM equals to 1 this would denote an absolute similarity between a reference and an encoded videos, i.e., the higher SSIM value denotes a better resulting video quality. It should be noted that as videos' resolution increases (videos 1-3 vs 4-6, Table 3), the SSIM difference between the hardware and the software encoding increases as well.

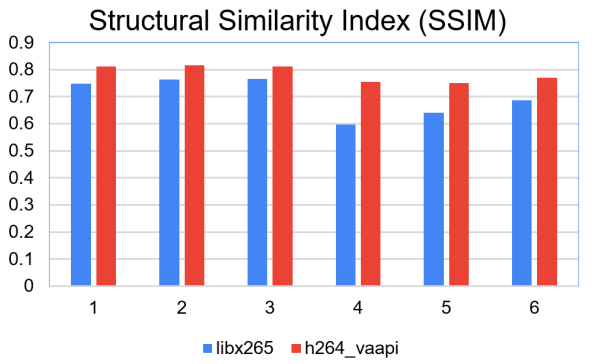

Fig. 5 Average SSIM values for 6 hardware (red) and software (blue) encoded video sequences. The Y-axis of SSIM is non-dimensional.

Figure 6 displays the average VMAF values that were

computed for the six videos. The X-axis shows a video ID, from 1 to 6. The Y-axis shows the average VMAF value (calculated with the Netflix library) corresponding to each video ID and is non-dimensional. In contrary to the PSNR and the SSIM, the VMAF also takes into an account temporal characteristics of an input video (Table 3, last column).VMAF values for software encoding were slightly higher than for hardware encoding, while the higher values should be preferred.

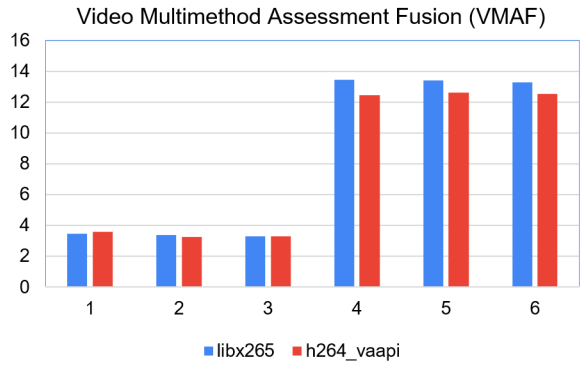

Fig. 6 Average VMAF values for 6 hardware (red) and software (blue) encoded video sequences. The Y-axis of VMAF is non-dimensional.

Figures 7, 8, and 9 demonstrate an average CPU, RAM and time usage accordingly, for each video with the software and the hardware encoding. In each case the software encoding used 3-4 times more CPU than the hardware encoding. Moreover, the hardware encoding used about 5 times less RAM in average, each time having stable and relatively small values, while the software encoding demonstrated different RAM usage for every input video. In addition, in each case the software encoding consumed about 6 times more time than the hardware encoding.

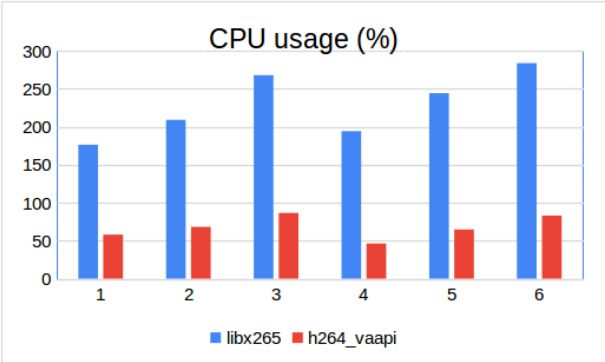

Fig. 7 Average 4 cores CPU usage of the hardware and the software encoding for 6 input videos.

5. CONCLUSION AND FUTURE WORK

This paper presented a comparison of video encoding for the hardware encoding with FFmpeg h264_vaapi and the software encoding with libx265 encoders. We implemented encoding in C++ using FFmpeg libraries and encoded 6 different videos with the hardware and the software approaches, 10 times for each video. The resulting

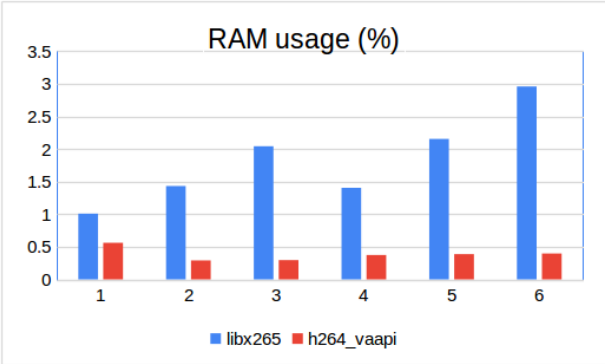

Fig. 8 Average RAM usage of the hardware and the software encoding for 6 input videos.

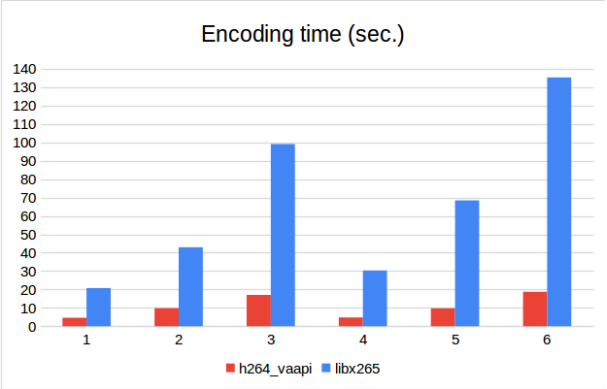

Fig. 9 Average time of the hardware and the software encoding for 6 input videos.

quality of encoding was compared using PSNR, SSIM and VMAF metrics. The hardware encoding showed better results with regard to PSNR and SSIM metrics, but was slightly outperformed by the software encoding for VMAF metrics. The comparison demonstrated that the hardware encoding with h264_vaapi consumed about 6 times less encoding time and about 5 times less RAM than the software encoding with libx265. Moreover, CPU load of the hardware encoding was 3-4 times less than of the software encoding. At the same time, the resulting video quality of the hardware and the software encoding did not differ significantly. Therefore, if an encoding application is applied for mobile robotics tasks where the importance of real-time streaming ultimately dominates over insignificant differences in video frame (image) quality, the hardware encoding should be preferred over the software encoding.

As a part of our future work, we plan to add a hardware encoding module to our RTSP server for Servosila Engineer mobile robot [27]. Next, the new hardware encoding module performance will be compared with the existing software encoding module in order to confirm the findings of the current study using the target hardware and real world environment setup.

ACKNOWLEDGMENT

This work was supported by the Russian Foundation for Basic Research (RFBR), project ID 19-58-70002.

This work was partially supported by the research grant of Kazan Federal University. The fifth author acknowledges the support of the Japan Science and Technology Agency, the JST Strategic International Collaborative Research Program, Project No.18065977.

REFERENCES

[1] A. Vokhmintsev, M. Timchenko, and K. Yakovlev, "Simultaneous localization and mapping in unknown environment using dynamic matching of images and registration of point clouds," in *Int. Conf. on Industrial Engineering, Applications and Manufacturing*, 2016, pp. 1–6.

[2] C. Cao, Y. Huang, Y. Yang, L. Wang, Z. Wang, and T. Tan, "Feedback convolutional neural network for visual localization and segmentation," *Transactions on pattern analysis and machine intelligence*, vol. 41, no. 7, pp. 1627–1640, 2018.

[3] R. Lavrenov and E. Magid, "Towards heterogeneous robot team path planning: acquisition of multiple routes with a modified spline-based algorithm," in *MATEC Web of Conferences*, vol. 113, 2017, p. 02015.

[4] S. Zhang, Z. Fan, L. Chen, C. Qian, and X. Gao, "Realizing hardware accelerated avs video playback on the godson platform," in *Int. Conf. on Electronics, Circuits, and Systems*, 2013, pp. 177–180.

[5] N. Alishev, R. Lavrenov, K.-H. Hsia, K.-L. Su, and E. Magid, "Network failure detection and autonomous return algorithms for a crawler mobile robot navigation," in *11th International Conference on Developments in eSystems Engineering (DeSE)*, 2018, pp. 169–174.

[6] S. Hall, "Encoding/decoding," *Media and cultural studies: Keyworks*, vol. 2, 2001.

[7] D. Ivanko, A. Karpov, D. Ryumin, I. Kipyatkova, A. Saveliev, V. Budkov, D. Ivanko, and M. Železnỳ, "Using a high-speed video camera for robust audio-visual speech recognition in acoustically noisy conditions," in *International Conference on Speech and Computer*, 2017, pp. 757–766.

[8] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h. 264/avc video coding standard," *Transactions on circuits and systems for video technology*, vol. 13, pp. 560–576, 2003.

[9] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *Transactions on circuits and systems for video technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[10] R. Safin, R. Lavrenov, T. Tsoy, M. Svinin, and E. Magid, "Real-time video server implementation for a mobile robot," in *11th International Conference on Developments in eSystems Engineering (DeSE)*, 2018, pp. 180–185.

[11] I. Mavrin, R. Lavrenov, and E. Magid, "Development of a graphical user interface for a crawler mobile robot servosila engineer," in *11th International Conference on Developments in eSystems Engineering (DeSE)*, 2018, pp. 192–197.

[12] M. H. Schimek, B. Dirks, H. Verkuil, and M. Rubli, "Video for linux two api specification," *History*, vol. 6, p. 11, 1999.

[13] J. De Cock, A. Mavlankar, A. Moorthy, and A. Aaron, "A large-scale video codec comparison of x264, x265 and libvpx for practical vod applications," in *Applications of Digital Image Processing XXXIX*, vol. 9971, 2016, p. 997116.

[14] J. Kufa and T. Kratochvil, "Software and hardware hevc encoding," in *Int. Conf. on Systems, Signals and Image Processing*, 2017, pp. 1–5.

[15] R. Bundulis and G. Arnicans, "Use of h. 264 real-time video encoding to reduce display wall system bandwidth consumption," in *3rd Workshop on Advances in Information, Electronic and Electrical Engineering*, 2015, pp. 1–6.

[16] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. Sullivan, "Rate-constrained coder control and comparison of video coding standards," *Circuits and Systems for Video Technology*, vol. 13, pp. 688 – 703, 2003.

[17] A. Hore and D. Ziou, "Image quality metrics: Psnr vs. ssim," in *Int. Conf. on Pattern Recognition*, 2010, pp. 2366–2369.

[18] R. Rassool, "Vmaf reproducibility: Validating a perceptual practical video quality metric," in *International Symposium on Broadband Multimedia Systems and Broadcasting*, 2017, pp. 1–2.

[19] S. T. Welstead, *Fractal and wavelet image compression techniques*, 1999, vol. 40.

[20] A. Hore and D. Ziou, "Image quality metrics: Psnr vs. ssim," in *Int. Conf. on Pattern Recognition*, 2010, pp. 2366–2369.

[21] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *The 37th Asilomar Conference on Signals, Systems & Computers*, vol. 2, 2003, pp. 1398–1402.

[22] "Vmaf - video multi-method assessment fusion," https://github.com/Netflix/vmaf, 2020.

[23] K. E. Psannis, "Hevc in wireless environments," *Journal of Real-Time Image Processing*, vol. 12, no. 2, pp. 509–516, 2016.

[24] C. Angelini, "Intel's second-gen core cpus: The sandy bridge review," https://tomshardware.com/reviews/sandy-bridge-core-i7-2600k-core-i5-2500k,2833-5.html, 2011.

[25] Github - netflix/vmaf: Perceptual video quality assessment based on multi-method fusion. [Online]. Available: https://github.com/Netflix/vmaf

[26] atop - linux man page. [Online]. Available: https://linux.die.net/man/1/atop

[27] I. Moskvin, R. Lavrenov, E. Magid, and M. Svinin, "Modelling a crawler robot using wheels as pseudo-tracks: Model complexity vs performance," in *7th International Conference on Industrial Engineering and Applications (ICIEA)*, 2020, pp. 235–239.