Real-time Video Server Implementation for a Mobile Robot

Safin Ramil
Intelligent Robotic Systems Laboratory
Kazan Federal University
Kazan, Russia
safin.ramil@it.kfu.ru

Roman Lavrenov
Intelligent Robotic Systems Laboratory
Kazan Federal University
Kazan, Russia
lavrenov@it.kfu.ru

Tatyana Tsoy
Intelligent Robotic Systems Laboratory
Kazan Federal University
Kazan, Russia
tt@it.kfu.ru

Mikhail Svinin
College of Information Science and Engineering, Ritsumeikan University
Kyoto, Japan
svinin@fc.ritsumei.ac.jp

Evgeni Magid
Intelligent Robotic Systems Laboratory
Kazan Federal University
Kazan, Russia
magid@it.kfu.ru

***Abstract*— Most of robots are using vision for various applications. In some cases, mobile robots are provided with an insufficient onboard processing hardware, and therefore video from cameras needs to be transmitted in an efficient and reliable way to a more powerful system for further off-board processing. Multiple difficulties could be faced during video streaming software development, including high latencies, network congestion, packet losses, distortions and others, which makes trade-offs between video quality, bitrate, frame rate, and packet loss inevitable. Thus, the key problem is to find such parameters, which will satisfy the specified needs. In our work we implement a video streaming server on mobile robot Servosila Engineer. A set of experiments demonstrated that high bitrates and frame rates increase load on CPU. Packet losses could be mitigated by decreasing bitrate to 100-200 kbps.**

Keywords—mobile robot, algorithm, wireless network, video streaming, engineering

I. INTRODUCTION

Many applications of robotics employ visual information in order to perform various tasks including visual Simultaneous Localization and Mapping (SLAM) [1][2], teleoperation and path planning, obstacle avoidance, and human-robot interaction [3]. In case of mobile robots, equipped with one or more cameras, particularly, stereo vision system, the issue of a system's poor performance may arise, thus, requiring data to be efficiently transmitted over wireless network to a more powerful remote computer. Some applications may need a real-time video information, e.g. teleoperation in urban search and rescue (USAR), especially, in highly dynamic environments. Real-time video streaming imposes additional requirements on the network, especially, regarding time delays.

Wireless networks provide mobility to a robot, allowing to move freely within some limited area. However, wireless signals are subject to interferences, attenuation and, generally, wireless communication channels have less bandwidth compared to wired ones, which requires to compress data. After the compression stage, encoded video units are transmitted using one of transport layer protocols - Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP guarantees that units of data, that are encapsulated in packets, will be reliably delivered in a correct order, which adds extra delays. On the contrary, UDP, does not deal with retransmission or data integrity issues, leaving this functionality to higher levels of abstraction (e.g., application). On a higher level of abstraction, there are different streaming protocols, which are designed to deliver multimedia content (both video and audio) in a most reliable and fast way enabling to handle error-checking and retransmission issues by introducing specific mechanisms, e.g., a sequence number to keep track of order, checksum, etc.

In our work we employ Russian crawler-type mobile robot Servosila Engineer equipped with four cameras [4]. It is designed to be utilized in USAR operations, demining, and firefighting. Those tasks require a teleoperator to operatively response to a particular situation and to be aware of dynamically changing environment. Robot's original client-server application programming interface (API) enables to receive high latency video stream through an ad-hoc wireless network. Also, only one camera at a time can be used with a possibility to switch to another one manually. Given the network and hardware capabilities of the existing mobile robot, the key challenge is to find an optimal video streaming solution. In our previous work we calibrated robot's cameras and developed a software to capture raw video stream [5]. In this paper we present implementation of a video streaming server.

The rest of the paper is organized as follows. Section 2 describes mobile robot hardware and software characteristics. Section 3 reviews video quality assessment. Section 4 presents a short review of video coding standards. Section 5 reviews real-time streaming protocol. Related work on video streaming system development is described in Section 6. We present experimental work in Section 7 and conclude in Section 8.

978-1-5386-6712-5/18/$31.00 ©2018 IEEE
DOI 10.1109/DeSE.2018.00042

II. Hardware and Software of Setup

A. Hardware Capabilities

Crawler-type mobile robot Servosila Engineer (Fig. 1) is equipped with a dual core (physical cores) Intel Core i7-3517UE CPU working at 1.7 – 2.8 GHz and 4GB DDR3 RAM. Even though there is no dedicated GPU on this machine, an integrated Intel HD Graphics 4000 can handle some image processing tasks, e.g., image format conversion, resampling, etc.

Robot vision system uses four cameras, which are located in a way that provides a good situation awareness. Three cameras are installed on the front side, while the fourth camera is located on the rear side of the robot's head (Fig. 2). Depending on a camera, its raw image format can be either 8-bit Bayer pattern RGB (regular cameras) or YUV (for the camera with an optical zoom). Camera-related features are presented in Table 1.

Wireless connection throughput between the robot and a teleoperator (a main computer) at the distance of 2-3 m without any obstacles resulted in 2.5 Mbit/s. Measurement were obtained with a maximum transmission unit (MTU) network parameter set to 1500 bytes using *ipef3* tool.

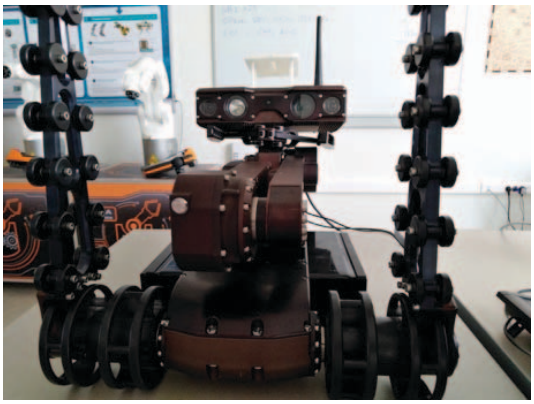

Fig. 1. Servosila Engineer crawler-type mobile robot (front view)

Fig. 2. Servosila Engineer robot's front with three front view cameras (on the left) and a rare view camera (on the right)

TABLE I. Vison System Features of Servosila Engineer Robot

Camera	Features		
	Supported pixel format	***Width x Height***	***Framerate***
Rare view, left and right front cameras	YUYV 4:2:2	1280 x 720	50
Front view central camera	8-bit Bayer RGB (GRGR/BGBG)	640 x 480	15, 25, 30, and 60
		744 x 480	

B. System and Software

Robot's on-board computer operates under Ubuntu 14.04.5 LTS (Trusty Tahr) operating system. Manufacturer's client-server API enables to obtain a compressed video stream from the installed cameras with UDP transport protocol. Each datagram carries a video frame encoded as a JPEG image. It is possible to disable video compression and deliver raw image data instead. The original manufacturer's API is restricted to stream only from a single selected camera at a time. In order to change current video source to another one, a special control packet should be sent. Despite the fact that images are compressed and transmitted with UDP, high latencies of over 1.5 s were detected.

III. Video Quality Assessment

In order to provide clients with a satisfactory content, video streaming system should be developed with regard to Quality of Service (QoS) and Quality of Experience (QoE). QoS considers network statistics as well as coding parameters, e.g., delays, packet loss, jitter, bitrate, frame rate, etc. QoE is considered to be user-centric, and thus it is more subjective than QoS.

QoS, generally, does not always explicitly determine a video stream quality. Higher bitrates as well as higher frame rates can positively affect the quality of the video. However, the impact of a frame rate may be not as significant as the impact of a bitrate [6]. The frame rate affects the perceived video quality depending on the spatial and temporal characteristics of the video [7]. Higher frame rates for a given fixed bitrate leads to degradation of video quality. For increasing packet losses, the impairment effect can be mitigated by reducing the encoding bitrate [8].

IV. Video Coding Standards Overview

There are several video coding standards/schemes which enable to reduce data flow size while applying different algorithms, e.g., entropy coding, motion prediction, etc**.**

Nowadays, H.264 or MPEG-4 Part 10, Advanced Video Coding (H.264/AVC) is the most spread block-oriented video coding standard that was developed by ITU-T Video Coding Experts Group (VCEG) along with ISO/IEC JTC1 Moving Picture Experts Group (MPEG). MPEG-H Part 2, or High Efficiency Video Coding (HEVC), is an emerging video codec standard of a joint partnership of ITU-T VCEG and ISO/IEC MPEG standardization organizations. Primary goals of this standard are bitrate savings while keeping the same perceptual video quality (rate-distortion characteristics), putting emphasis on high-resolution video compression (e.g., 4K or even 8K), and support for more sophisticated parallel processing capabilities (e.g., tiles and wavefront parallel processing) comparing to its predecessor, H.264/AVC. VP9 is another royalty-free video coding format (scheme) that was developed by Google. It is explicitly configured for a high-resolution content. However, it is applicable for lower resolutions as well.

Nearly two-fold bitrate savings are achieved using HEVC reference software relative to H.264/AVC for the same objective quality (peak signal-to-noise ratio, PSNR), although, HEVC demands more computational power [9]. Furthermore, informal subjective quality comparison confirms that HEVC enables to gain a substantial improvement in compression capability for low-delay applications while maintaining the same perceptual

video quality [10]. Both subjective and objective rate-distortion performance analysis exhibits the average bitrate reduction of HEVC relative to H.264/AVC (39.6% for objective based measurements and 52.6% for subjective scores) and VP9 (35.6% - objective and 49.4% - subjective) [11]. However, another objective large-scale video codec compression test results, which included *x264* (H.264/AVC), *x265* (HEVC) and libvpx (VP9) encoder implementations, demonstrate that not only *x265* but *libvpx* can lead to significantly lower data rates over *x264* [12]. At a lower resolution (e.g., 360p) *x265* appeared to be more efficient than *libvpx* (30.8% and 22.6% comparing to the *x264*). Although, at higher resolution (e.g., 1080p) the gap between them decreases (43.4% and 43.5% vs. *x264*). Considering vulnerable for data loss wireless environments HEVC was more stable for various packet loss rates than H.264/AVC [13]. Also, HEVC is shown to be especially effective for low bitrates and low-delay communication scenarios [14].

Literature review shows superiority of HEVC over H.264/AVC and VP9, especially, in case of low video resolutions, substantially improving rate-distortion characteristics, at the expense of growing computational complexity, which can be potentially mitigated.

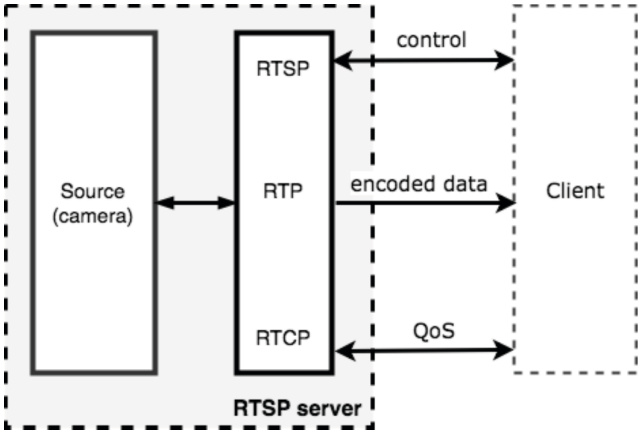

Fig. 3. RTSP, RTP, and RTCP multimedia streaming protocols workflow

V. Real-Time Streaming Protocol

Encoder receives video stream and issues compressed data units that are ready to be transmitted over an existing network. However, some delivery issues may arise, for example, such as packet loss, synchronization of multiple multimedia streams, out-of-order packets arrival. To overcome those issues multimedia-streaming protocols were developed. Real Time Streaming Protocol (RTSP) is a stateful application layer protocol dedicated for the needs of controllable real-time multimedia delivery over IP networks. It enables to establish and control sessions using commands as in HTTP (e.g., play, pause, teardown) that are sent through TCP-based connection while multimedia data flows via other channels, e.g. UDP (Fig. 3).

RTP is a transport protocol responsible for real-time end-to-end media delivery. It guarantees neither delivery nor prevent out-of-order data arrival. However, possible issues may be handled on behalf of a receiver with the help of provided instruments, e.g., sequence number, timestamp. Typically, RTCP serves as a complementary monitoring and synchronization protocol designed to give feedback on transmitted data quality (QoS).

There are a several C++ RTP libraries and frameworks that enable creating client-server video streaming applications. For example, *JRTPLIB* is an easy to use object-oriented RTP library allowing to send/receive multimedia streams over RTP (SRTP, which is a secure RTP, is also supported) and internally handle RTCP functionality. By contrast, *Live555* represents a more compound framework comprising of libraries for RTP, RTCP, RTSP based multimedia streaming. It is an event-driven server system with its own scheduling and event-loop mechanisms.

VI. Streaming Server Implementation

In our RTSP server implementation we employed the following instruments:

- *FFmpeg* (version 3.4.2)
- *Live555 Framework* (version 2017.10.28)
- *Video4Linux* (version 2)
- *Boost* (version 1.54.0)
- *log4cpp*
- *libx265* (version 2.5.2)

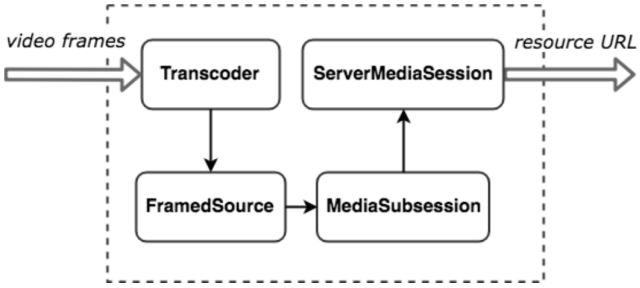

Fig. 4. Main components of our RTSP server implementation based on Live555 Framework (multimedia resource represented by Transocder can be accessed at the specified URL)

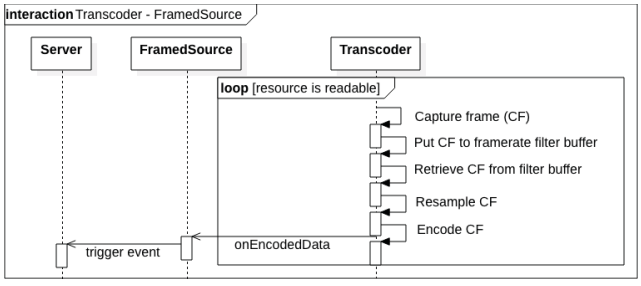

Fig. 5. Simplified sequence diagram of the interaction between FramedSource, Transcoder, and Live555 server (in case of new encoded data is available specified event is triggered)

Main components of the server are illustrated in Fig. 4. Transcoder captures (decodes) and encodes raw video stream coming from the camera applying some modifications (e.g., resampling, frame rate reduction). Then, encoded data units are

passed to FramedSource component, which serves as an intermediate layer between the camera and Live555 server due to the event-driven system architecture. Media session may contain one or more subsessions, for example, for encoded video and audio streams. Video encoding and decoding is accomplished using *FFmpeg*, particularly, we used *libx265* for encoding and *Video4Linux* for decoding process. After raw video frames are captured from the camera, they go through the framerate reduction, resampling, and conversion steps – filtering (Fig. 5, 6).

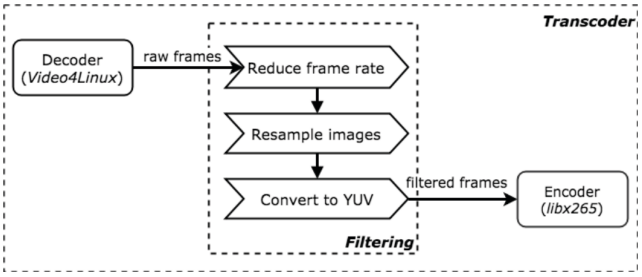

Fig. 6. Video stream preprocessing flow (frame rate reduction, resampling, and image format conversion steps)

VII. Experiments

In order to examine and find an optimal solution for real-time video streaming from the Servosila Engineer mobile robot's cameras, we conducted experiments on the implemented RTSP server using stereo vision system of the robot. The RTSP server was built into an executable binary file by *gcc* compiler (version 4.8.4). Video streaming server configuration options were passed as command-line arguments handled by *Boost* library (Table 2). Some of encoder-specific configuration parameters are shown in Table 3. Experiments were comprised of a set of coding and network parameters such as image resolution, frame rate, bitrate, and datagram size (Table 4). In total we conducted 216 experiments.

Ubuntu 14.04 LTS Linux computer with quad-core CPU and 8 GB RAM served as a client to the video streaming server. In order to estimate QoS, on the client side the *openRTSP* program was installed. CPU and memory usage statistics of the streaming server were estimated using *pidstat* utility. We implemented TCP-based client-server wrapper in Python programming language for test evaluation purposes. Interaction process between the client and the server is shown in Fig. 7. Initially, TCP connection was established between the client (a remote computer) and the server (the robot). After the RTSP server was launched with specified parameters, it sent a message to the client indicating that video server is ready to accept connections. On the remote computer *openRTSP* was launched to estimate QoS statistics. Finally, after designated time was over, the RTSP server process was terminated (sending SIGKILL).

Both client and server information were saved in separate log files. However, the log files are not convenient to be utilized. Thus, a log parser was implemented, which retrieves all necessary information into a more readable form using regular expressions.

TABLE II. RTSP Server's Command-line Options

Option	Description	Default value
--udp-size	UDP datagram size (bytes)	1500
--out-width	width of the resampled frames	640
--out-height	height of the resampled frames	480
--out-fps	encoder's output frame rate (used by *FFmpeg* fps filter)	4
--bitrate	desired output bitrate provided by the encoder (kbps)	100
--vbv-bufsize	bitrate control buffer size (Kb)	200

TABLE III. HEVC Encoder Parameters (libx265)

Parameter	Value
width	specified by the command-line parameter frame width value (--out-width)
height	specified by the command-line parameter frame height value (--out-height)
profile	HEVC Main profile
framerate	specified by the command-line parameter frame rate value (--out-fps)
pixel_format	yuv420p (YUV Planar with 4:2:0 chroma sampling)
preset	ultrafast
tune	zerolatency
vbv-maxrate	specified by the command-line parameter bitrate value (--bitrate)
vbv-bufsize	specified by the command-line parameter bufsize value in bytes (--vbv-bufsize)

TABLE IV. Streaming Parameters for Experiment Evaluation (216 trials is conducted)

Parameter	Value
resolution (width x height)	240 x 180 (low)
	480 x 360 (average)
	640 x 480 (high)
frame rate (fps)	4, 7, 10, 12
bitrate (kbps)	100, 200, 300, 400, 600, 800
datagram size (bytes)	500, 1500, 2200

VIII. Results

The 216 experiments results were stored in three log files for cameras and server information, containing the following network and encoding information: video frames width and height; a frame rate (fps); a bitrate (kbps); a datagram size (bytes); packets loss percentage; an average quantization parameter (QP); CPU and memory usage. When it comes to streaming video from the mobile robot over wireless network, the most challenging issues are packet losses and huge CPU load. Robot's CPU processes all heavy tasks related to image processing, video capturing and encoding. Depending on the resolution, CPU usage considerably changes in value reaching two-fold or one and a half times growth in case of resolution switching from low to average and form average to high, respectively. Streaming bitrate affects not only the resulting

quality of the video, but also affects CPU. Given fixed frame rate and resolution, for the set of bitrate values starting form 100 kbps up to 800 kbps, load on CPU consistently increases (Fig. 8, 9).

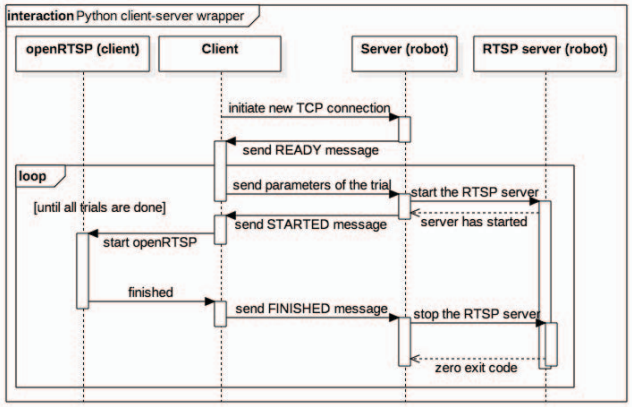

Fig. 7. Sequence diagram of the conducted experiment evaluation (client-server wrapper interaction including RTSP server lifecycle control and QoS statistics estimation using *openRTSP*)

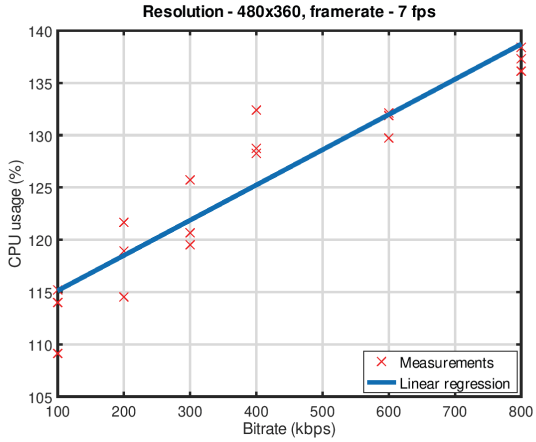

Fig. 8. CPU usage and bitrate linear regression (frame resolution 480x360 and frame rate 7 fps, CPU usage changes from 115.2% to 138.4%)

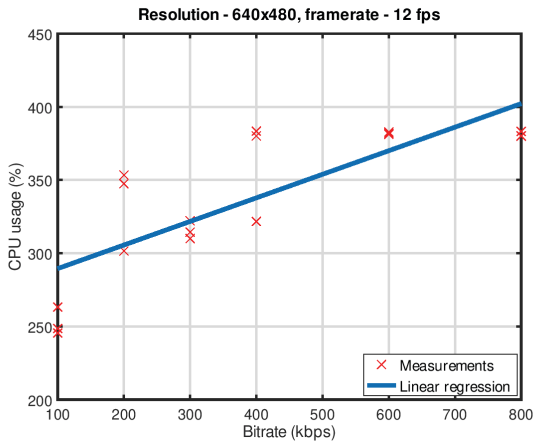

Fig. 9. CPU usage and bitrate linear regression (frame resolution 640x480 and frame rate 12 fps, CPU usage changes from 245.47% to 380.27%)

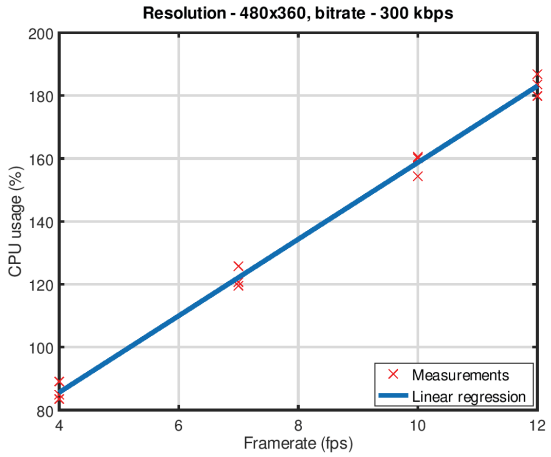

Fig. 10. CPU usage and frame rate linear regression (frame resolution 480x360 and bitrate 300 kbps, CPU usage increases from 89.07% to 183.53%)

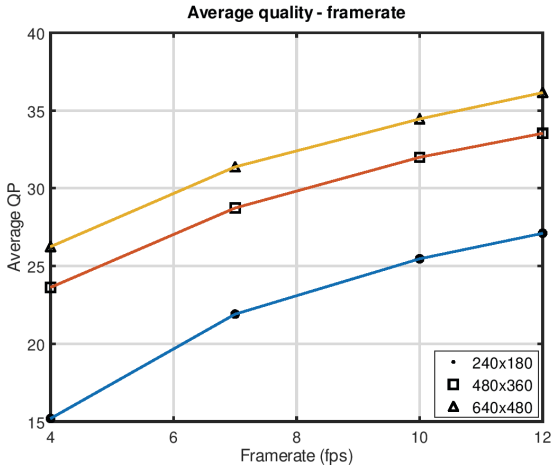

Fig. 11. Frame rate influence on average video quality represented by the QP values at the given bitrate of 300 kbps; accumulation of measurements on the same frame rate level indicates different frame resolutions: low (blue), average (red), high (orange)

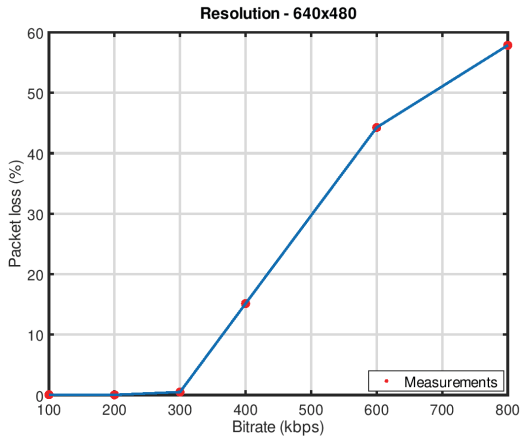

Fig. 12. Packet losses for the bitrate values from 100 kbps up to 800 kbps at 640x480 frame resolution

When it comes to a frame rate, higher values also require more computational power (Fig. 10). Even though, a frame rate is not as significant as a bitrate, it influences user experience, especially, in highly dynamic video scenes. It should be noted that at a fixed bitrate value a frame rate appeared to affect average video quality. As it is demonstrated in Fig. 11, video quality degrades as frame rate increases. Accumulation of QP values in three different levels on the same frame rate value indicates low, average and high frame resolutions. The higher the frame resolution, the less quality it has at the given bitrate.

Packet losses during video stream delivery through wireless network may occur spontaneously, e.g., due to interference. Figures 12, 13 demonstrate packet losses for different frame resolutions. At 640x480 significant packet losses occurred after 300 kbps. Yet, the packet losses were smaller when datagram size was 500 bytes. At the lowest resolution of 240x180 the packet losses started to grow after 200 kbps.

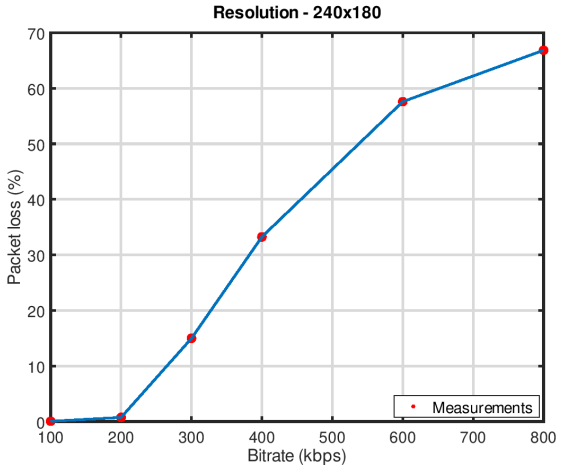

Fig. 13. Packet losses for the bitrate values from 100 kbps to 800 kbps at 240x180 frame resolution

IX. Conclusions

In this paper we presented our real-time video streaming server implementation for mobile robot Servosila Engineer. The server is based on RTP/UDP and RTSP protocols and employs HEVC encoder enabling to keep the same visual quality while significantly decreasing video size. Wireless networks have much less bandwidth, more influenced by interferences, therefore, making video streaming more complicated. In order to find out, which parameters affect video streaming, experiments were conducted. We examined a bitrate, a frame rate, a datagram size, a frame resolution, and packet losses parameters. The experiments were automated and the results were collected into log files. The results showed that for a robot with no dedicated GPU it is hard to process video efficiently. Higher bitrates, frame rates, and frame resolutions require significant computational power. In case of higher frame rates, an average video quality is decreasing and a load on CPU is dramatically increasing. Therefore, lower frame rates are preferred to be utilized in case of operation on non-highly dynamic environments. The key issue of wireless data transmission is packet losses. Experiments demonstrated that only low bitrates could be employed for achieving low packet losses, i.e. 100 – 200 kbps.

Acknowledgment

This work was supported by the Russian Foundation for Basic Research (RFBR), project ID 18-58-45017. Part of the work was performed according to the Russian Government Program of Competitive Growth of Kazan Federal University.

References

[1] Buyval A., Afanasyev I., and Magid E., “Comparative analysis of ROS-based monocular SLAM methods for indoor navigation,” Proc. of SPIE - The Int. Society for Optical Engineering, vol. 10341. p. 103411, 2017.

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

[3] Magid E. and Takashi T., "Static Balance for Rescue Robot Navigation Translation Motion Discretization Issue within Random Step Environment," In ICINCO-2010, vol. 2, pp. 415-422, 2010.

[4] Safin R., Lavrenov R., Saha S. K., and Magid E., "Experiments on mobile robot stereo vision system calibration under hardware imperfection," MATEC Web of Conf., vol. 161, p. 03020, 2018.

[5] Safin R., and Lavrenov R., “Implementation of ROS package for simultaneous video streaming from several different cameras,” Int. Conf. on Art. Life and Robotics, pp. 220-223, 2018.

[6] Khan A., Sun L., and Ifeachor E., "QoE prediction model and its application in video quality adaptation over UMTS networks," IEEE Trans. on Mult., vol. 14(2), pp. 431-442, 2012.

[7] Huynh-Thu Q., and Ghanbari M., "Temporal aspect of perceived quality in mobile video broadcasting," IEEE Trans. on Broad., vol. 54(3), pp. 641-651, 2008.

[8] Boumehrez F., Brai R., Doghmane N., and Mansouri K., “Quality of experience enhancement of high efficiency video coding video streaming in wireless packet networks using multiple description coding,” J. of Electron. Im., vol. 27(1), p. 013028, 2018.

[9] Dissanayake M. B., and Abeyrathna D. L. B., "Performance Comparison of HEVC and H. 264/AVC Standards in Broadcasting Environments," J. of Inf. Proc. Sys., vol. 11(3), 2015.

[10] Horowitz M., Kossentini F., Mahdi N., Xu S., Guermazi H., Tmar H., and Xu J., “Informal subjective quality comparison of video compression performance of the HEVC and H. 264/MPEG–4 AVC standards for low-delay applications,” App. of Digital Im. Proc. XXXV, vol. 8499. Int. Soc. for Opt. and Phot.: 2012, p. 84990W.

[11] Řeřábek M., and Ebrahimi T., "Comparison of compression efficiency between HEVC/H.265 and VP9 based on subjective assessments," App. of Digital Im. Proc. XXXVII, vol. 9217. Int. Soc. for Opt. and Phot.: 2014, p. 92170U.

[12] De Cock J., Mavlankar A., Moorthy A., and Aaron A., “A large-scale video codec comparison of x264, x265 and libvpx for practical VOD applications,” App. of Digital Im. Proc. XXXIX, vol 9971. Int. Soc. for Opt. and Phot.: September 2016, p. 997116.

[13] Psannis K. E., "HEVC in wireless environments," J. of Real–Time Im. Proc., vol. 12(2), pp. 509–516, 2016.

[14] Ohm J. R., Sullivan G. J., Schwarz H., Tan T. K., and Wiegand T., “Comparison of the coding efficiency of video coding standards – including high efficiency video coding (HEVC),” IEEE Trans. on circuits and sys. for video tech., vol. 22(12), pp. 1669–1684, 2012.