Implementation of ROS package for simultaneous video streaming from several different cameras

Ramil Safin, Roman Lavrenov
Intelligent Robotic Systems Laboratory, Higher Institute for Information Technology and Information Systems (ITIS), Kazan Federal University, 35 Kremlyovskaya street
Kazan, 420008, Russian Federation
E-mail: safin.ramil@it.kfu.ru, lavrenov@it.kfu.ru
http://kpfu.ru/robolab.html

Abstract

Real time video stream capturing and processing is important for a variety of tasks in robotics. We created ROS package that captures concurrent video stream from 4 different cameras of Russian mobile robot "Servosila" Engineer. V4L2 API was used to configure video devices and to retrieve raw data from cameras. Memory mapping approach of mapping device buffers increased overall performance by eliminating redundant memory copies. We demonstrate the comparison of our new package and OpenCV based package.

Keywords: Video Streaming, V4L2, ROS, C++11, OpenCV, mobile ground robot, experiments.

1. Introduction

Capturing video stream from a robot is an important task for multiple purposes, e.g., visual simultaneous localization and mapping (SLAM)[1] in path planning[2], human-robot interaction[3], teleoperation in urban search and rescue[4], etc. In some cases, it is necessary to retrieve and process video data in a real-time mode. Moreover, while single cameras are used for monocular SLAM algorithms[5] for UAVs and simple robots, for more sophisticated robots multiple cameras could be used, for example, as a stereo pair in order to implement SLAM. Thus, as these algorithms may request large computing powers a mobile robot may require transferring sensory data to a more powerful computing device for information processing and analysis.

In our ROS package development process, we used Russian crawler robot Servosila Engineer[6]. It has four cameras and client–server application programming interface (API). Even though it is possible to switch between the cameras using original GUI of the robot, it cannot stream video data from all cameras at one time. Therefore, we created robot operating system (ROS) package that enables streaming video from the four cameras simultaneously. Our next goal is to stream real time video data via a wireless connection and to develop a server based on a real-time transport protocol (RTP) that will enable clients to receive video data from the robot and improve teleoperation process as an operator can receive more information about environment.

2. Vision-related features of the robot

A crawler-type mobile robot Engineer (Fig.1) is equipped with four cameras that provide good situation awareness. Three of the four cameras in the robot head are located on the front side and one is a rear view camera:

- frontal optical zoom camera
- frontal wide-angle cameras pair (stereo vision)
- wide-angle rear view camera

The installed operating system (OS) is Ubuntu 14.04 LTS (Trusty Tahr). The CPU is Intel® Core™ i7-3517UE (1.70GHz) with 2 physical cores.

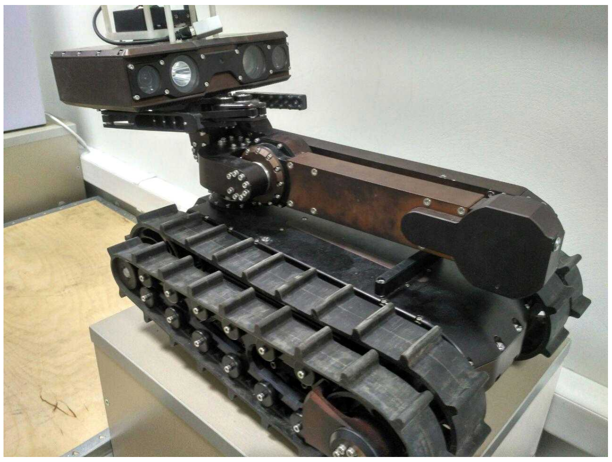


Fig. 1. Servosila Engineer crawler-type UGV

3. Video streaming ROS package

3.1. *Streaming with V4L2 API*

To capture video from the cameras, we use the second version of Video4Linux API (V4L2 API), which is pre-installed within the OS by the maker. Programming a V4L2 video device consists of the following steps[7]:

- Opening the device
- Changing device properties
- Selecting a video standard
- Negotiating a data format
- Negotiating an input/output method
- The actual input/output loop
- Closing the device

Video device parameters are adjusted by input/output control (ioctl) requests. It is a system call for manipulating underlying device parameters (i.e. video format, frames per second, etc.).

The workflow of typical V4L2 application (Fig. 2) may vary depending on use case. For example, our implementation does not render captured frames. The initial step is to open the device as it is needed in order to be able to adjust device for streaming. In order to invoke ioctl requests, device's file descriptor should be opened for reading and writing operations.

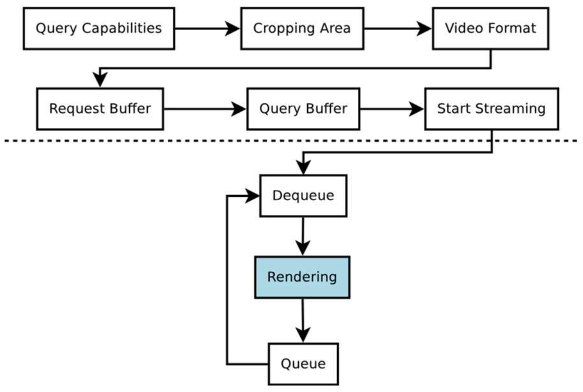


Fig. 2. Common V4L2 video capturing application workflow

Secondly, device capabilities are queried. The purpose of this step is to ensure whether device is able to stream and capture video. It is necessary to negotiate exchanged data (image) format in order to avoid ambiguity. In our case, Engineer's three cameras (stereo pair and rare view) support only Bayer pattern format[8] as a raw data from the device's driver. The optical zoom camera's raw data presents an image encoded in YUV color space.

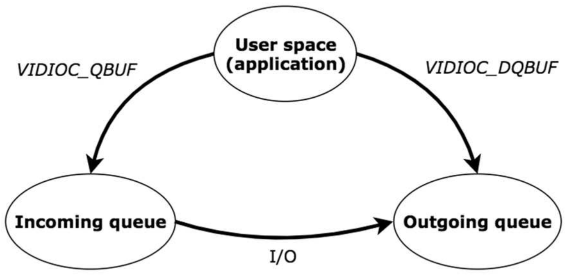


Fig. 3. The process of retrieving frames from V4L2 device using special V4L2 API calls (placing buffer in the incoming queue and getting raw data from outgoing queue)

After negotiating video streaming data format, device buffers are allocated. A buffer contains raw image data exchanged by the application and device's driver using streaming input/output (I/O) methods. In order to handle captured video data, memory mapping (mmap) I/O approach is used. It increases performance by eliminating redundant memory copies from user to kernel space. The application and the driver exchange pointers to buffers. Hence, we can access captured video data directly in application memory.

Then, we need to query buffers in order to obtain information about allocated buffers, such as memory location (pointers) and buffer length (size).

The process of getting recent frames consists of the queuing and dequeuing parts (Fig. 3). Special API calls are invoked (VIDIOC_QBUF and VIDIOC_DQBUF). Queue operation puts the buffer in the driver's incoming queue. The buffer will be waiting for the driver to fill it with data. Dequeue operation is used to retrieve processed buffer (with video data) from the outgoing queue[9].

3.2. *ROS package implementation*

Originally, no ROS distribution was installed on the robot's OS. Therefore, we installed Ubuntu 14.04 compatible ROS Indigo distribution. The architecture of the package consists of the following main parts (Fig. 4):

- ROS publisher node. Each camera independently publishes its raw image data.
- V4L2 API layer. All low-level interactions with video devices are handled by V4L2 API.

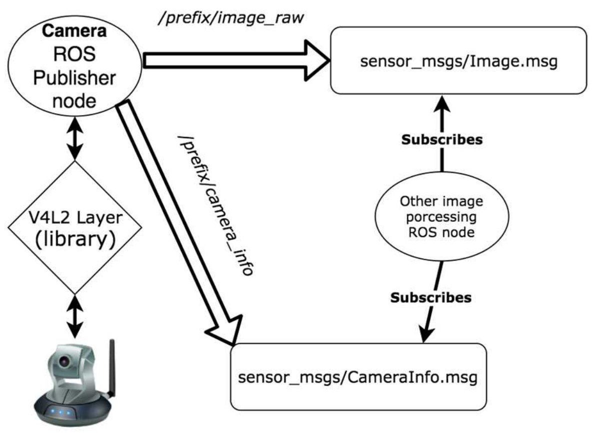
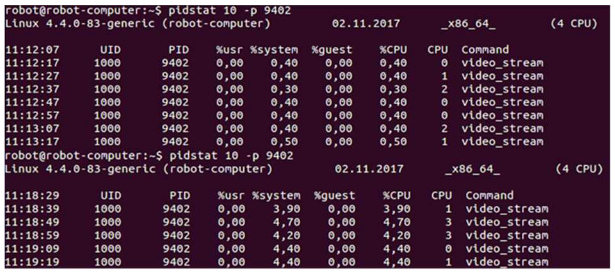


Fig. 4. Our ROS video streaming package architecture (ROS publisher node, V4L2 API layer, raw image and camera information topics)

The ROS publisher node streams video data from a selected camera to the assigned topic. A subscribed node could be a server or some image processing node. Also, the following node's streaming parameters are used:

- Frames per second (fps). Video frames publishing rate.
- Image resolution.
- ROS image format (i.e. "bayer_grbg8").
- Device path (i.e. "/dev/video0").



```
robot@robot-computer:~$ pidstat 10 -p 9402
Linux 4.4.0-83-generic (robot-computer)       02.11.2017      _x86_64_        (4 CPU)

11:12:07      UID       PID    %usr %system  %guest    %CPU   CPU  Command
11:12:17     1000      9402    0,00    0,40    0,00    0,40     0  video_stream
11:12:27     1000      9402    0,00    0,40    0,00    0,40     1  video_stream
11:12:37     1000      9402    0,00    0,30    0,00    0,30     2  video_stream
11:12:47     1000      9402    0,00    0,40    0,00    0,40     0  video_stream
11:12:57     1000      9402    0,00    0,40    0,00    0,40     0  video_stream
11:13:07     1000      9402    0,00    0,40    0,00    0,40     2  video_stream
11:13:17     1000      9402    0,00    0,50    0,00    0,50     1  video_stream
robot@robot-computer:~$ pidstat 10 -p 9402
Linux 4.4.0-83-generic (robot-computer)       02.11.2017      _x86_64_        (4 CPU)

11:18:29      UID       PID    %usr %system  %guest    %CPU   CPU  Command
11:18:39     1000      9402    0,00    3,90    0,00    3,90     1  video_stream
11:18:49     1000      9402    0,00    4,70    0,00    4,70     3  video_stream
11:18:59     1000      9402    0,00    4,20    0,00    4,20     3  video_stream
11:19:09     1000      9402    0,00    4,40    0,00    4,40     0  video_stream
11:19:19     1000      9402    0,00    4,40    0,00    4,40     1  video_stream
```

Fig. 5. Our ROS video streaming package's CPU usage statistics (measured by the *pidstat* utility in the idle and non-idle modes)

Captured image is distorted, therefore, camera calibration is an essential part of our future work plans.

At the development stage, several issues were faced. The most important one consisted in the inability to determine video frame's captured time, which are important for stereo SLAM algorithms implementation. V4L2 API gives the ability to determine frame's captured time by means of captured buffer structure's parameter. Originally, this parameter had an invalid state, and a proper solution was obtained from the OpenCV GitHub repository[10]. The timestamp data of the buffer structure should be extracted after dequeuing and before the next queueing stage.

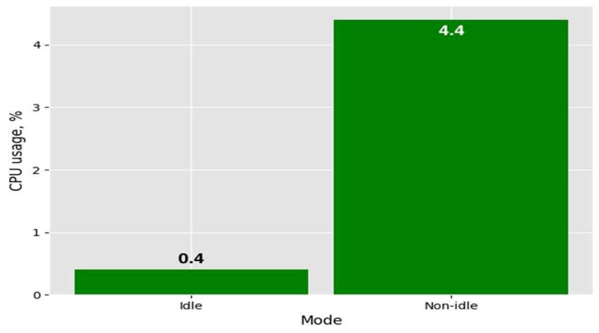


Fig. 6. Average CPU usage of our ROS video streaming package (in the idle and non-idle modes)

All further experiments conducted on Servosila Engineer's OS.

The created ROS video streaming package shows encouraging results (Fig. 5, 6). Note, that if there are no subscribers to the video stream package's topic CPU usage tends to zero. Otherwise, if there is at least one subscriber, then our package consumes approximately 4% of CPU resources. It should be mentioned that no image data conversion is performed, and video data is published as a raw data by means of creating ROS image message. Hence, we can decrease precious CPU usage on a robot's system, not wasting resource on image conversion. Our package does not depend on the OpenCV.

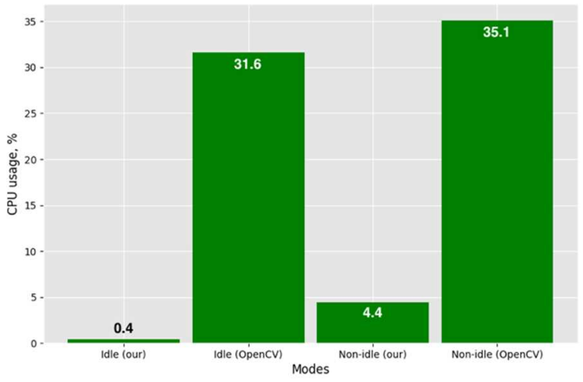


Fig. 7. Average CPU usage of OpenCV based ROS package in comparison with our implementation (in the idle and non-idle modes) (using *pidstat* utility)

We reviewed other ROS video streaming packages. The most interesting one seemed to be an OpenCV based package[11]. Comparing CPU usage of this package to our implementation we concluded that OpenCV based one uses considerably more computational resources (Fig. 7). If we stream from more cameras, then CPU usage will increase and computational power will not be enough for other applications. Also, it is important to note that in idle mode, when there are no subscribers to the streaming node, OpenCV based package continues to consume a lot of CPU resources.

4. Conclusions and future plans

Streaming video from multiple cameras is an important task. The key point is not only to get frames from camera but also to decrease the usage of system resources. In this paper, we introduced video streaming ROS package implementation. This package demonstrated relatively low consumption of system resources in either idle or non-idle modes. As our long-term goal, we plan to develop an RTP server, which will enable to stream video data from several cameras in a real-time mode via wireless connection. Our source code is available for public access on the GitHub version control system[12].

Acknowledgements

This work was partially supported by the Russian Foundation for Basic Research (RFBR) and Ministry of Science Technology & Space State of Israel (joint project ID 15-57-06010). Part of the work was performed according to the Russian Government Program of Competitive Growth of Kazan Federal University.

References

1. J. Fuentes-Pacheco, J. Ruiz-Ascencio and J. M. Rendón-Mancha, Visual simultaneous localization and mapping: a survey, *Artificial Intelligence Review* (2015), v 43(1), pp. 55-81.
2. E. Magid, R. Lavrenov and A. Khasianov, Modified spline-based path planning for autonomous ground vehicle, *Int. Conf. on Informatics in Control, Automation and Robotics* (2017), pp.132-141.
3. A. Karpov, S. Carbini, A. Ronzhin and J. Viallet, Two similar different speech and gestures multimodal interfaces, *Multimodal User Interfaces* (2008), pp. 155-184.
4. E. Magid and T. Tsubouchi, Static balance for rescue robot navigation: Discretizing rotational motion within random step environment, *Int. Conf. on Simulation, Modeling, and Programming for Autonomous Robots* (Springer, Berlin, Heidelberg, 2010), pp. 423-435.
5. A. Buyval, I. Afanasyev and E. Magid, Comparative analysis of ROS-based Monocular SLAM methods for indoor navigation, *9th Int. Conf. on Machine Vision* (2017), pp. 103411K-103411K-6.
6. M. Sokolov, R. Lavrenov, A. Gabdullin, I. Afanasyev and E. Magid, 3D modelling and simulation of a crawler robot in ROS/Gazebo, *4th Int. Conf. on Control, Mechatronics and Automation* (2016), pp. 61-65.
7. L. Yinli, Y. Hongli, and Zh. Pengpeng. The implementation of embedded image acquisition based on V4L2, *IEEE Int. Conf. on Electronics, Communications and Control* (2011), pp. 549-552.
8. A. Lukin and D. Kubasov, High-quality algorithm for Bayer pattern interpolation. *Programming and Computer Software 30.6* (2004), pp. 347-358.
9. V4L2 manual; https://linuxtv.org/downloads/v4l-dvb-apis/uapi/v4l/v4l2.html
10. OpenCV GitHub Issues, timestamp issue; https://github.com/opencv/opencv/issues/8763
11. GitHub video_stream_opencv ROS package: https://github.com/ros-drivers/video_stream_opencv
12. LIRS video streaming ROS package; https://github.com/chupakabra1996/lirs_ros_video_streaming