Journal of
open research software

Shakir, D I et al 2017 GIFT-Grab: Real-time C++ and Python Multi-channel Video Capture, Processing and Encoding API. *Journal of Open Research Software,* 5: 27, DOI: https://doi.org/10.5334/jors.169

SOFTWARE METAPAPER

GIFT-Grab: Real-time C++ and Python Multi-channel Video Capture, Processing and Encoding API

Dzhoshkun Ismail Shakir[1,2], Luis Carlos García-Peraza-Herrera[1], Pankaj Daga[1], Tom Doel[1,2], Matthew J. Clarkson[1], Sébastien Ourselin[1] and Tom Vercauteren[1]

[1] Translational Imaging Group, CMIC, Department of Medical Physics and Biomedical Engineering, University College London, GB

[2] University College London Hospitals NHS Foundation Trust, GB

Corresponding author: Dzhoshkun Ismail Shakir, Research Associate, Honorary Research Fellow (d.shakir@ucl.ac.uk)

GIFT-Grab is an open-source API for acquiring, processing and encoding video streams in real time. GIFT-Grab supports video acquisition using various frame-grabber hardware as well as from standard-compliant network streams and video files. The current GIFT-Grab release allows for multi-channel video acquisition and encoding at the maximum frame rate of supported hardware – 60 frames per second (fps). GIFT-Grab builds on well-established highly configurable multimedia libraries including FFmpeg and OpenCV. GIFT-Grab exposes a simplified high-level API, aimed at facilitating integration into client applications with minimal coding effort. The core implementation of GIFT-Grab is in C++11. GIFT-Grab also features a Python API compatible with the widely used scientific computing packages NumPy and SciPy.

GIFT-Grab was developed for capturing multiple simultaneous intra-operative video streams from medical imaging devices. Yet due to the ubiquity of video processing in research, GIFT-Grab can be used in many other areas. GIFT-Grab is hosted and managed on the software repository of the Centre for Medical Image Computing (CMIC) at University College London, and is also mirrored on GitHub. In addition it is available for installation from the Python Package Index (PyPI) via the pip installation tool.

Keywords: multi-channel; video capture; real-time video encoding; video processing; hardware-accelerated video encoding; GPU; frame-grabber hardware; network video streaming; Epiphan DVI2PCIe Duo; Blackmagic DeckLink SDI 4K; Epiphan Pearl; C++; Python; NumPy; SciPy; Xvid; H.265; H265; VP9; HEVC

Funding statement: This work was supported through an Innovative Engineering for Health award by the Wellcome Trust [WT101957], the Engineering and Physical Sciences Research Council (EPSRC) [NS/A000027/1] and a National Institute for Health Research Biomedical Research Centre UCLH/UCL High Impact Initiative. Sébastien Ourselin receives funding from the EPSRC (EP/H046410/1, EP/J020990/1, EP/K005278) and the MRC (MR/J01107X/1). Luis C. García-Peraza-Herrera is supported by the EPSRC-funded UCL Centre for Doctoral Training in Medical Imaging (EP/L016478/1).

(1) Overview

Introduction

GIFT-Grab is an open-source application programming interface (API) for acquiring, processing and encoding video data in real time. GIFT-Grab supports live video acquisition via supported frame-grabber hardware as well as the capture of standard-compliant network streams [1]. GIFT-Grab also supports offline video acquisition from video files. GIFT-Grab is implemented in C++11 yet it also comes with a Python API that facilitates video data processing with NumPy [2] and SciPy [3], two popular Python scientific computing packages widely used in academia. GIFT-Grab can be built from source code and can also be installed from the Python Package Index (PyPI) [4] using the pip installation tool [5] (please see the installation note in the **Availability** section).

GIFT-Grab leverages the video capture and processing functionality of external multimedia libraries including FFmpeg [6] and OpenCV [7], and of software development kits (SDKs) provided by frame-grabber hardware manufacturers. Although FFmpeg comes with powerful capabilities, it is architected in a low-level C-language coding style. Due to this, it requires unintuitive configuration involving numerous parameters, even for seemingly simple tasks. On the other hand, OpenCV provides a higher-level API than FFmpeg, resulting in a shallower learning curve. However it does not support callbacks; but instead requires the client application to query video sources for new data.

Video data acquired from external sources is frequently used in an image processing pipeline. It becomes a challenge for the end user to convert between various datatypes

representing video data in different software packages, especially when using various capabilities offered by different libraries and SDKs. It is not uncommon for different video processing libraries to have no common data interface other than a byte array. On the other hand, scientific computing packages like SciPy [3] provide powerful image processing capabilities in conjunction with more natural datatypes like the NumPy array [2]. GIFT-Grab aims to bridge this gap of video acquisition, processing and subsequent encoding, by providing a pipeline-oriented architecture which facilitates real-time video processing with minimal coding effort. We have developed GIFT-Grab to meet the requirements of the international research initiative "Guided Instrumentation for Fetal Therapy and Surgery" ("GIFT-Surg") [8]. GIFT-Surg involves major innovations in science and engineering combined with clinical translation for improving fetal therapy and diagnosis by providing advanced image processing and visualisation capabilities. We have already obtained promising results with a number of novel methods including real-time mosaicing of the placenta [9] and sensor-free real-time instrument tracking [10]. These methods rely on real-time video streams from medical devices such as endoscopes and ultrasound probes. GIFT-Grab seamlessly makes video data from medical devices available for use in these medical imaging methods. It supports the simultaneous acquisition of multi-channel video streams at up to 60 Full HD (1080p) fps.

An integral part of the development cycle of new medical imaging methods is validation with data representative of the targeted medical scenario. In the case of GIFT-Surg this requires making intra-operative video data from medical devices available for offline use. However raw video data imposes an impractical burden in terms of storage requirements. For instance one Full HD (1920 × 1080) video frame with three channels occupies approximately 6 megabytes (MB) in raw format. At a frame rate of 60 fps, storing an hour's worth of intra-operative endoscopic video requires 1.25 terabytes (TB) of storage space. It is clear that permanently storing raw video is not a scalable solution. However it is also easy to see that even temporarily storing raw video (e.g. for offline encoding) would push storage hardware to its limit if not handled carefully. To remedy this, GIFT-Grab supports real-time encoding of video in popular formats including High Efficiency Video Coding (HEVC) [11] and Xvid [12]. GIFT-Grab leverages the real-time data processing capabilities of graphics processing units (GPU) for streamlining intra-operative video capture. GIFT-Grab is made available to the community as open-source under a permissive licence [1, 4].

Implementation and architecture

Before diving into the details of the GIFT-Grab architecture, listing 1 illustrates a simple video processing pipeline using the GIFT-Grab Python API. As an example, this pipeline includes saving Gaussian-smoothed video data captured from a frame-grabber card to a file. The reader should note that the source code in listing 1 is trimmed for brevity. A full working example is provided in listing 3.

```
# Connect to an Epiphan DVI2PCIe Duo device
source_fac = VideoSourceFactory.get_instance()
source = source_fac.get_device(
    Device.DVI2PCIeDuo_DVI, ColourSpace.BGRA
)

# Get a Gaussian smoother
gauss = GaussianSmootherBGRA()

# Get a file writer
target_fac = VideoTargetFactory.get_instance()
target = target_fac.create_file_writer(
    Codec.HEVC, "/tmp/myfile.mp4",
    source.get_frame_rate()
)

# Attach the Gaussian smoother to the live video
stream
source.attach(gauss)
# Attach the file writer to the Gaussian smoother
gauss.attach(target)

# [...]
#
# Frames automatically processed in the
background
#
# [...]

# Detach the Gaussian smoother from the live
video stream
source.detach(gauss)
# Detach the file writer from the Gaussian
smoother
gauss.detach(target)

# Device object automatically deallocated by
source_fac
```

Listing 1: Python code snippet demonstrating a GIFT-Grab pipeline for capturing live video data from an Epiphan DVI2PCIe Duo card [13]; processing and encoding it for saving to a video file. Note that the Python imports (six lines of code) have been excluded for brevity. The processing node of the pipeline is a class called `GaussianSmootherBGRA` and defined in listing 2. A full working example is provided in listing 3.

GIFT-Grab uses the abstract factory design pattern [14] as an abstraction layer representing the creation of connections to supported devices. Upon request, the video source factory creates a polymorphic object that implements the abstract video source interface (`IVideoSource`). This is illustrated in the UML class diagram in **Figure 1**. The video source factory manages each device connection as a singleton [14]. As a safety net against potential resource issues, it also takes care of properly destroying all created device connection singletons at the end of its lifetime.

GIFT-Grab relates video producers (sources) to video consumers (targets) via the observer design pattern [14]. The observer design pattern is a high-level design paradigm similar in concept to function callbacks. It allows observers to subscribe to video sources (i.e. to be attached), so as to be notified of each new video frame. The observable-observer hierarchy in GIFT-Grab is illustrated by the UML class diagram in **Figure 2**.

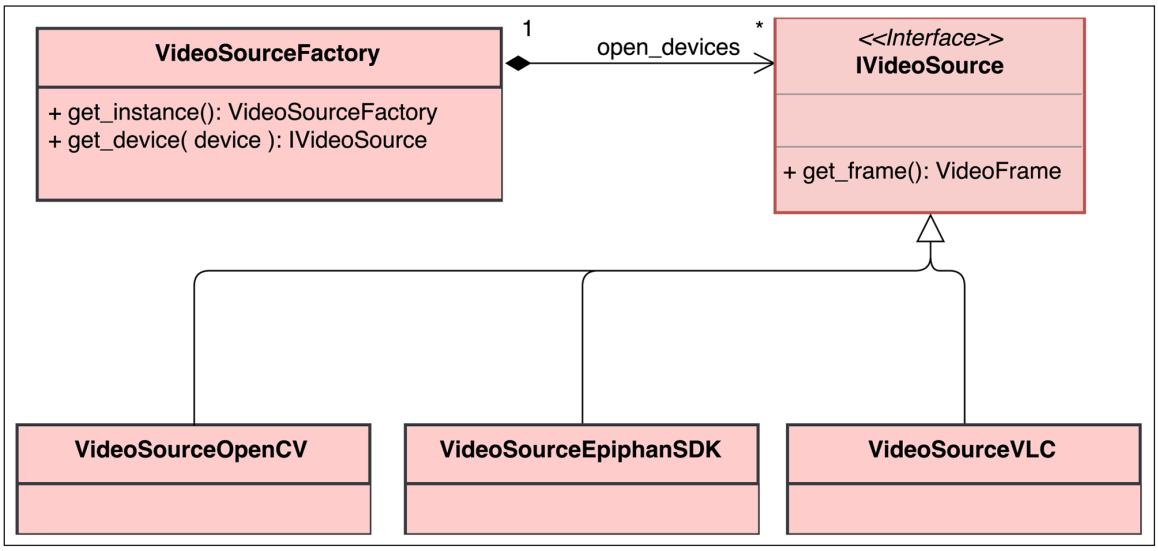

Figure 1: UML class diagram showing how connections to supported devices are created and maintained in GIFT-Grab. The `VideoSourceFactory` singleton creates on demand a polymorphic `IVideoSource` object that serves as the interface for acquiring video frames from a supported device. The diagram shows three of the derived classes which implement this interface. The choice of the `IVideoSource` type depends on the connection request.

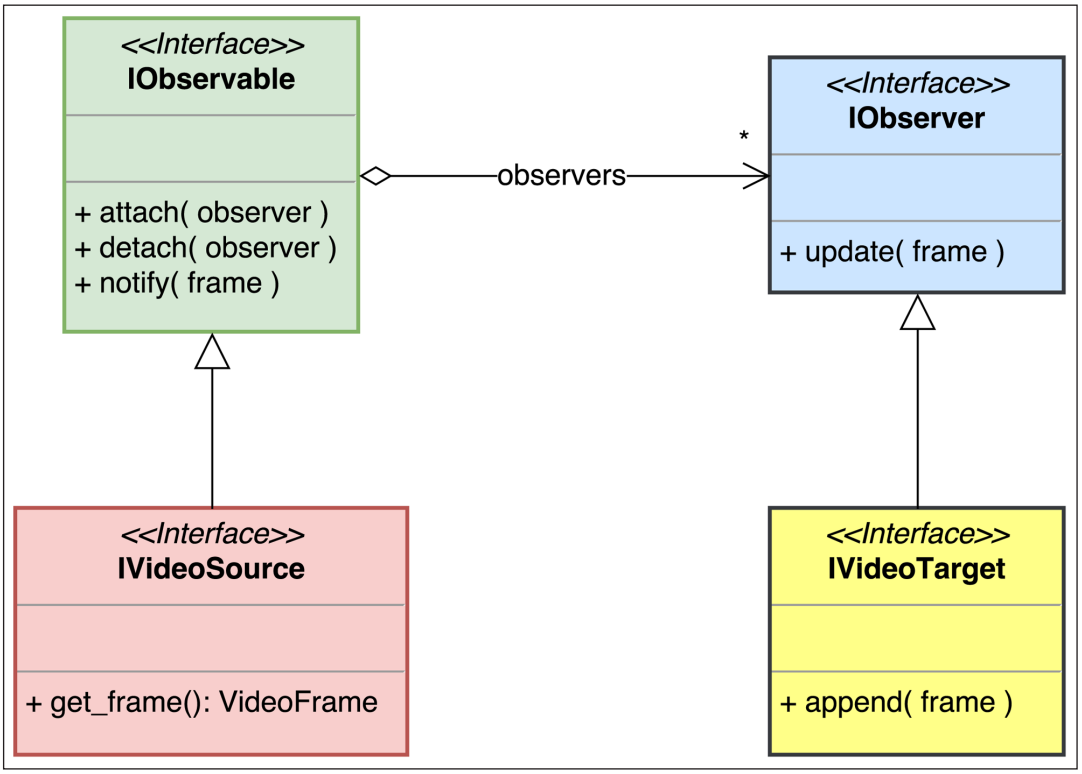

Figure 2: UML class diagram showing the observer design pattern hierarchy for video sources and targets in GIFT-Grab. Every `IVideoSource` is also an `IObservable`, to which `IObservers` can be attached. Each time a new `VideoFrame` becomes available, the `IObservable` notifies all its `IObservers` by calling the `update` method. Similarly, each `IVideoTarget` is an `IObservers`, with its `update` method automatically calling the `append` method (i.e. appending that particular frame to the file).

The observable-observer hierarchy is a solid building block that facilitates video processing pipelines in GIFT-Grab. The exemplary processing pipeline shown in listing 1 includes a processing node. This node is the `GaussianSmootherBGRA` class detailed in listing 2 that smoothes arriving frames using the GIFT-Grab NumPy [2] data wrapper in conjunction with SciPy image processing functions [3], and subsequently passes them further down the pipeline.

```
from pygiftgrab import IObservableObserver
import scipy.ndimage as ndimage

class GaussianSmootherBGRA(IObservableObserver):

    def __init__(self):
        super(GaussianSmootherBGRA, self).__init__()

    def update(self, frame):
        data_np = frame.data(True)
        ndimage.gaussian_filter(
            input=data_np, sigma=(5, 15, 0),
            order=0, output=data_np)
```

Listing 2: Python code snippet showing a sample video processing node: upon receiving a video frame `GaussianSmootherBGRA` smoothes it with a Gaussian kernel and feeds it further down the processing pipeline (such as the one shown in listing 1) – cf. **Figure 2**. GIFT-Grab wraps video data into NumPy arrays [2] to allow for processing with SciPy routines [3]. Note that the sole purpose of this code snippet is to demonstrate GIFT-Grab's compatibility with NumPy and SciPy. As such `GaussianSmootherBGRA` does not implement any data buffering (such as a ring buffer). Data buffering might be required to enable such a processing node to cope with real-time frame rates depending on the capabilities of the platform hardware.

The GIFT-Grab `IVideoSource` implementors make use of functionality provided by external libraries such as OpenCV [7] and Epiphan Video Grabber SDK [15] to realise the actual data connection to the supported devices. Some of these libraries inherently support callbacks, thereby making the implementation of the observer design pattern intuitive. However others operate on a data-querying basis rather than providing a callback mechanism. In

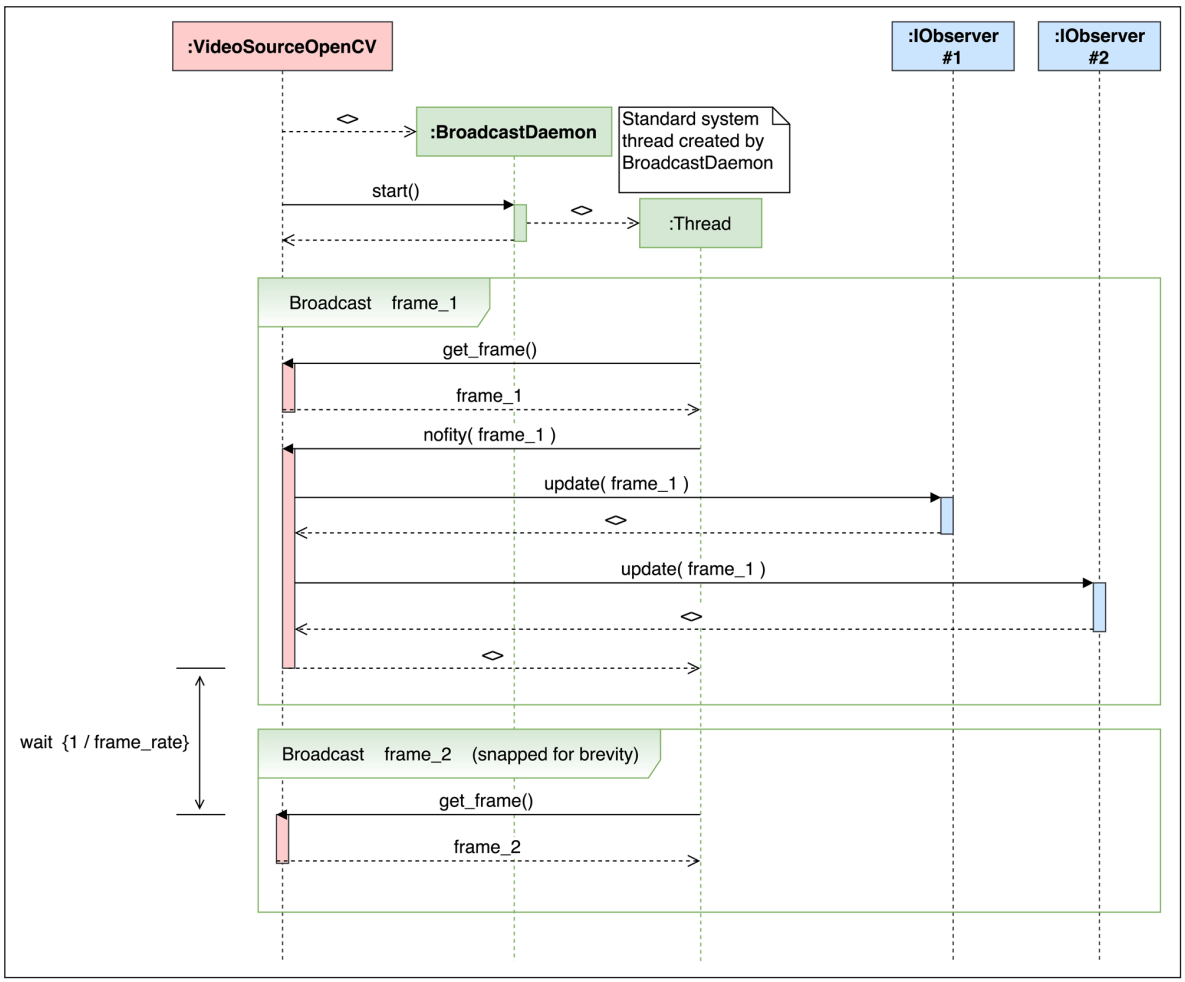

Figure 3: UML sequence diagram showing how the visitor design pattern [14] is used for realising the observer pattern with an `IVideoSource` implementor that does not inherently support callbacks: The `BroadcastDaemon` visitor queries the `VideoSourceOpenCV` object in regular intervals for a new `VideoFrame`. Once obtained, the new `VideoFrame` is propagated to all attached observers by calling `notify`.

GIFT-Grab we remedy this issue by using the visitor design pattern [14]: a daemon object acts as the visitor to any `IVideoSource` implementor that does not inherently support the observer paradigm. This daemon object queries the video source in regular time intervals for new video frames, and subsequently notifies all observers of that video source. This is shown in **Figure 3**.

GIFT-Grab uses the abstract factory design pattern [14] also for abstracting the creation of video targets (see **Figure 4**). Video targets abstract the encoding of video frames to video files. The `VideoTargetFactory` operates on the "resource acquisition is initialisation" (RAII) principle [16]: each created `IVideoTarget` is immediately ready for use (see listing 1). As shown in **Figure 2** each video target is also an observer in the observable-observer hierarchy, which allows it to be attached to a video source for automatically saving each frame from a connected device. The abstract factory design pattern helps minimise the exposed API, which allows for a better encapsulation of the underlying implementation details.

Real-time encoding benchmarks

We benchmarked the real-time performance of GIFT-Grab by measuring the time it takes to encode video frames to a video file using a GPU. The video frames were read from the "Big Buck Bunny" sequence [17], in two different colour spaces: I420 and BGRA. They were then used for producing three different HEVC-encoded MP4 video files [11, 18] of different resolutions:

- **4K (3840 × 2160):** Native resolution of the "Big Buck Bunny" sequence
- **Full HD (1920 × 1080):** All frames cropped to the middle 1920 × 1080 region
- **HD (1280 × 720):** All frames cropped to the middle 1280 × 720 region

We measured the wall-clock execution time of the `append` function of `VideoTargetFFmpeg` using the Boost Timer Library [19]. This execution time includes copying the video data from the CPU to the GPU, encoding the frame and actually writing it to the target video file. In

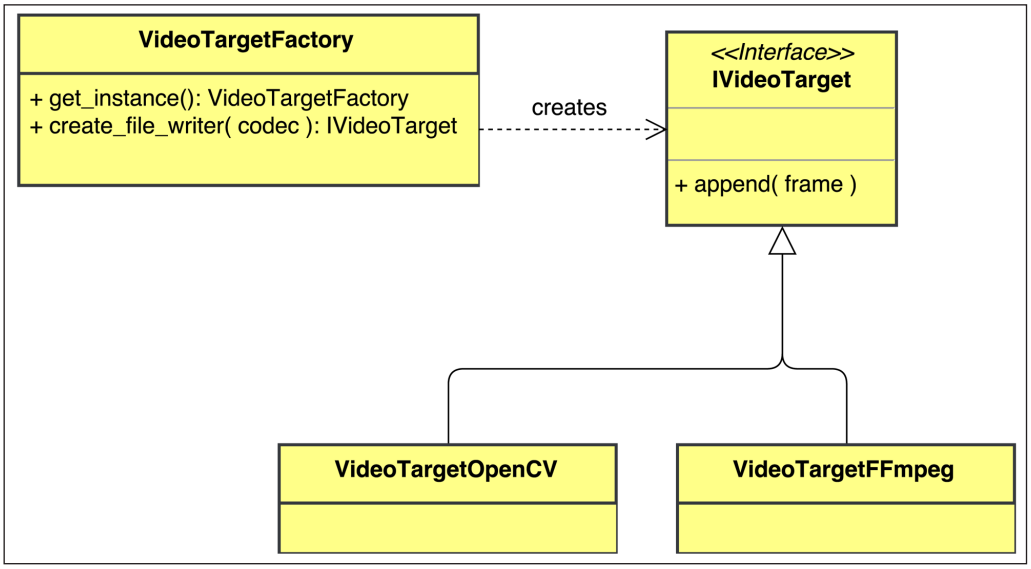

Figure 4: UML class diagram representing the video target hierarchy in GIFT-Grab. Two implementors of the `IVideoTarget` interface are shown: `VideoTargetOpenCV` and `VideoTargetFFmpeg` classes which are used for encoding video frames and saving them to video files. Similar to the case of `VideoSourceFactory`, the `VideoTargetFactory` singleton creates on demand a polymorphic `IVideoTarget` object. However contrary to the case of `VideoSourceFactory`, the ownership of the created `IVideoTarget` object passes to the caller, i.e. the caller is responsible for destroying it to free up memory at the end of its lifetime.

Table 1: Recorded maximum CPU load (as percentage) and maximum system memory used (in gigabytes) while recording each of the datasets shown in figure 5. Note that these are system-wide measurements to give an idea of the system load during the benchmarking process.

		GeForce GTX TITAN X	GeForce GTX 980 Ti
I420 frames	HD:	14.3%, 5.7 GB	14.3%, 5.7 GB
	Full HD:	14.7%, 5.7 GB	17.2%, 5.7 GB
	4K:	67.7%, 6.3 GB	75.8%, 6.3 GB
BGRA frames	HD:	18.5%, 3.8 GB	15.6%, 3.8 GB
	Full HD:	50.0%, 4.8 GB	16.7%, 4.8 GB
	4K:	52.9%, 11.0 GB	55.2%, 11.0 GB

addition we monitored the GPU memory used throughout the process, via the NVIDIA System Management Interface [20]. We report the maximum recorded GPU memory value for each case. Furthermore we monitored the total CPU load and the total system memory usage, using the Bash commands `top` and `free` respectively (see **Table 1**). We obtained these measurements on a workstation with two GPUs. At the time of recording, the workstation was in an idle state, with each of the GPUs running only a graphical user session. The hardware and software specifications of the workstation follow:

- **CPU:** Haswell FCLGA2011-3 with 6 Cores (12 Threads) 3.5 GHz (Intel Xeon E5-1650V3)
- **GPUs**
  1. **GeForce GTX TITAN X:** 3072 Maxwell Cores with Compute Capability 6.1 and 12 GB 7.0 Gbps GDDR5 memory (donated by the NVIDIA Corporation)
  2. **GeForce GTX 980 Ti:** 2816 Maxwell Cores with Compute Capability 5.2 and 6 GB 7.0 Gbps GDDR5 memory (ZOTAC GeForce GTX 980 Ti AMP!)
- **Memory:** 64 GB (4x 16 GB) DDR4-2133 quad-channel RAM (Crucial 64GB Kit)
- **Storage:** 500 GB SATA 6 Gbps SSD (Samsung 850 EVO)
- **Motherboard:** Asus X99-S
- **Operating system:** Ubuntu 16.04 LTS, kernel version: 4.4.0-72-generic
- **NVIDIA driver version:** 375.39
- **CUDA version:** 8.0

Figure 5 shows the wall-clock execution times for 100 video frames in each configuration. The execution time for the first video frame has been excluded from these graphs, but reported in **Table 2**. This is because the first frame is used for inferring the resolution to use in the resulting video file. This information is subsequently used for initialising the file writer object, including memory allocation.

The upper row in **Figure 5** demonstrates that in the absence of colour conversion (when I420 frames are fed to the file writer), all recorded execution times are well below 16.67 msec, the time interval between two video frames at 60 fps. The lower row shows that even when colour conversion is involved, 60 fps can still be achieved for Full HD video frames. However neither 60 fps nor 30

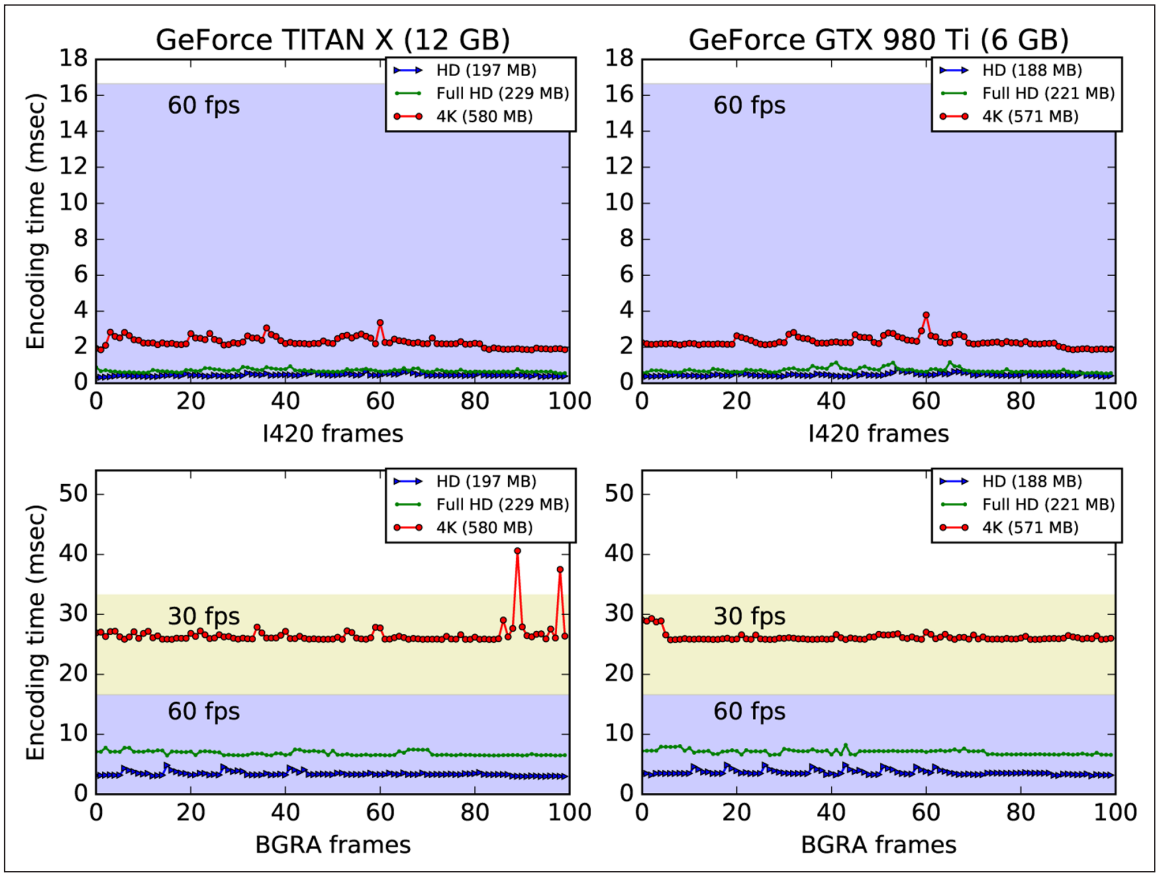

Figure 5: Wall-clock execution times in milliseconds for encoding 100 video frames of three different resolutions on two GPUs and subsequently writing them to a video file. Each column is for one GPU. The upper row shows execution times when I420 frames are fed into the file writer, i.e. no colour space conversion before encoding. The lower row shows the case when BGRA frames are fed into the file writer, i.e. BGRA-to-I420 conversion performed before encoding. The darker shade in each graph shows the region within the time limits that would allow for processing 60 fps. The lighter shade in the lower row shows the respective region for 30 fps (but not 60 fps). The numbers in paranthe-ses in the figure legends indicate the maximum GPU memory allocated at a time for each dataset.

Table 2: Encoding times in milliseconds for the excluded first frame (initialisation) in each of the datasets shown in figure 5.

		GeForce GTX TITAN X	GeForce GTX 980 Ti
I420 frames	HD:	486 msec	451 msec
	Full HD:	495 msec	412 msec
	4K:	754 msec	731 msec
BGRA frames	HD:	509 msec	441 msec
	Full HD:	505 msec	452 msec
	4K:	798 msec	736 msec

fps can be guaranteed for 4K video frames when colour conversion is involved. On the other hand, the relatively low GPU memory consumption suggests that multiple frames of the same video stream or frames of multiple video streams may be encoded in parallel. This could thus potentially yield higher frame rates when colour conversion is needed. Encoding multiple frames of the same video stream however requires proper synchronisation mechanisms to preserve the order of video frames, as well as proper handling of the inter-frame dependencies.

Quality control

GIFT-Grab can be built from source code or installed via the pip installation tool [5] from the Python Package Index (PyPI) [4]. We encourage the reader interested in the GIFT-Grab Python API to use this option (please see the installation note in the **Availability** section). Building GIFT-Grab from source requires configuration via CMake [21]. This installation option comes with a CMake discovery script that allows for seamlessly including GIFT-Grab in C++ software projects that use CMake [21].

Delivering high-quality sustainable medical imaging software is one of the core goals of GIFT-Surg [8]. Owing to the fact that the GIFT-Surg software deliverables are intended to be used for pre-operative surgical planning and intra-operative interventional guidance applications, we follow a development process that is inspired by the internationally accepted IEC EN 62304:2006 Standard "Medical device software – Software life cycle processes" [22]. This standard outlines the key requirements for the safe design and maintenance of software products intended for clinical use. We will briefly discuss the key sections of EN 62304 in relation to our development practices.

Section 5 of EN 62304 defines the requirements of the medical software development process. It emphasises traceability between each software development step and the software specifications, tests and risk control measures. It allows for an existing development process compatible with the standard to be adopted instead of defining a new one. We follow a development process very similar to that demonstrated by Höss et al. in a recent paper [23]. Höss et al. use state-of-the-art software development tools and technologies to implement the key requirements of EN 62304. As such they have not only achieved quality assurance (QA) of their software tool, but also laid a solid practical example for translating EN 62304 to the clinical ecosystem.

We use GitLab [24] for project management and Gitlab CI [25] for continuous integration (CI) in a similar fashion to how Höss et al. use Redmine [26] and Jenkins [27] respectively for these two tasks. The issue tracker functionality of GitLab serves to document feature requests and bug reports, which then steer the development process. GitLab allows for referencing issues from code commits, which ensures a direct link between the software specifications and the concrete development steps that lead to the realisation of those. This practice is also in line with section 6 of EN 62304, which outlines the requirements for maintaining clinical software.

We have adopted the test-driven development (TDD) process [28] for GIFT-Grab. In TDD each new feature or bug is first defined through a set of failing tests. Subsequently the implementation of the new feature or the bug fix boils down to adding new and/or refactoring the existing source code in a manner that makes the new tests pass. GitLab CI automatically executes the GIFT-Grab tests as well as the documented deployment procedures. This is a safety net that ensures changes to GIFT-Grab do not break existing functionality. In other words, when implementing a new feature or fixing a bug, only the code changes that make the new tests pass without causing the existing ones to fail are merged into the stable code branch. This practice is inspired by the requirement for identifying and resolving potential regressions during the active development of medical device software, as outlined in section 5 of EN 62304.

The GIFT-Grab tests are organised as a comprehensive test suite that is configured using CTest [29]. This test suite uses Catch [30] and pytest [31], at the C++ and Python levels respectively. The unit tests implemented using Catch [30] serve two purposes. First, they ensure that the video source and target factory instances are created according to the singleton pattern. Second, they check that the video source factory manages the connection to each supported device as a singleton. The unit tests implemented using pytest [31] check the correct operation of all documented GIFT-Grab features. All the tests pertaining to video acquisition from supported frame-grabber hardware are executed on dedicated test machines with the respective cards installed. Owing to the polymorphism that enables treating video files and frame-grabber

cards simply as video sources, the unit tests for files are designed following a similar pattern to the unit tests for frame-grabber hardware. Sample video files with known specs are also included as part of the testing infrastructure. These together with a provided internal Python module that makes use of FFmpeg utility applications [6] serve to ensure video de-/encoding capabilities are operational. In addition to the unit tests, the GIFT-Grab pytest suite also provides real-time integration tests that create image processing pipelines where intermediate processing nodes measure the acquisition frame rates and report in cases where the measured frame rates are lower than documented values. Last but not least, the GIFT-Grab test suite also checks the correct exposure of raw video data as NumPy arrays. For further technical details we refer the reader to the GIFT-Grab documentation [1, 4]. Apart from the automated tests, we also run manual tests, notably before each release, where a human observer runs a sample Python application related to a specific GIFT-Grab feature and (visually) assesses the result (e.g. a video file produced) for correctness.

Section 8 of EN 62304 defines the requirements for the configuration management of medical device software. This among others involves the version management of the "software of unknown provenance" (SOUP) in relation to the versions of developed software. In the case of GIFT-Grab, SOUP corresponds to the third party software libraries used by GIFT-Grab. We clearly document these software libraries in the GIFT-Grab documentation alongside their respective versions that GIFT-Grab has been tested with. GitLab further allows for documenting project-specific information using the Markdown language [32]. Using GitLab, information can be organised in the form of wiki pages and the Markdown language provides means for cross-referencing between different documentation resources. We use these GitLab features for documenting details related to the relevant SOUP libraries, such as detailed and custom installation instructions wherever applicable.

Section 9 of EN 62304 stipulates that problems related to medical device software are to be documented, classified for criticality and impact, and subsequently investigated. It further demands that users of the software be advised of existing problems. Our GitLab infrastructure serves to document reported problems in the form of issue tickets as detailed above. In addition, one or more labels related to an issue's criticality and impact can be assigned to each ticket. Last but not least, each ticket can be marked for resolution in a specific release. We have also open-sourced GIFT-Grab on the popular GitHub [1]. GitHub provides a similar issue tracking and documentation functionality, and has millions of users. We believe making GIFT-Grab thus available to a broader audience will accelerate the discovery (and potentially the resolution) of problems.

GIFT-Grab is not a full medical diagnostic or guidance tool, but one of the building blocks thereof. In other words, GIFT-Grab is a component that serves to make data from external medical devices available for processing in clinical software. As such, we focus on defining interfaces i) between GIFT-Grab and medical devices by supporting appropriate frame-grabber hardware, and ii) between GIFT-Grab and clinical software via an appropriate representation of video data to facilitate integration and compatibility with a broad range of available software tools. This is inspired by the requirement of EN 62304 for defining software inputs and outputs, and the interfaces between the software system and other systems (section 5.2.2). Although we endeavour to follow development practices compatible with EN 62304, we are also aware that compliance with EN 62304 requires a formal process for quality management and risk mitigation, as well as for usability engineering [23]. Furthermore, these processes are not only to be formalised and followed, but will also be subject to a formal independent audit.

(2) Availability

Installation note: The reader interested in installing GIFT-Grab via the pip installation tool should read the installation instructions in [33] before proceeding. Availability information for GIFT-Grab at the time of preparing this manuscript is listed below.

Operating system

Linux (tested on Ubuntu 14.04 LTS, Ubuntu 16.04 LTS, and Debian 9 "testing")

Programming language

C++11, Python

Additional system requirements

Currently supported frame-grabber hardware:

- Epiphan DVI2PCIe Duo [13]
- Blackmagic DeckLink SDI 4K [34]

Also an NVIDIA Maxwell 2nd generation GPU (or newer) [35] is needed for real-time video encoding.

Dependencies

The required GIFT-Grab dependencies are listed in **bold face** below. The requirement for each of the others depends on the desired GIFT-Grab features. As such, all features are inactive by default, to ensure the minimal number of dependencies for installing GIFT-Grab. We refer the reader to the GIFT-Grab documentation [1] for more detailed information, in particular the minimum acceptable versions of the listed dependencies.

At time of composing this manuscript GIFT-Grab uses stable versions of all the listed external dependencies except for libVLC. This is because some critical issues in the stable releases of libVLC have been fixed in recent release candidates.

- **A C++11 compatible compiler such as GCC [36]**
- **CMake [21]**
- Python [37]

- Boost.Python [38]
- pkg-config [39]
- OpenCV [7]
- x265 [40]
- libvpx [41]
- kvazaar [42]
- libVLC (VLC SDK) [43]
- NVIDIA Video Codec SDK (NVENC) [35]
- FFmpeg [6]
- zlib [44]
- Epiphan Video Grabber SDK [15]
- Blackmagic Desktop Video SDK [45]
- POSIX threads (pthreads) [46]
- pytest [31]
- Catch [30]
- doxygen [47]

List of contributors
The list of authors includes all contributors.

Software location
Archive
Name: Zenodo
Persistent identifier: https://doi.org/10.5281/zenodo.840633
Licence: BSD 3-Clause License
Publisher: Dzhoshkun Ismail Shakir
Version published: 1708 (August 2017 release)
Date published: 09/08/2017

Code repository
Name: GitHub
Persistent identifier: https://github.com/gift-surg/GIFT-Grab
Licence: BSD 3-Clause License
Date published: 09/08/2017

Language
English

(3) Reuse potential

Real-time and non-real-time video processing are ubiquitous in our age. Many applications in different domains use video data. The GIFT-Surg research initiative [8] develops novel medical imaging methods for fetal therapy and surgery, that rely on video from medical devices [10, 9]. Other examples include real-time stereo reconstruction [48] and probe tracking [49] in robotic surgery, surveillance endoscopies [50], real-time panorama image synthesis [51], vehicle surveillance [52] and content-based video identification [53].

GIFT-Grab allows real-time video processing applications to connect to external devices and to encode video data from these devices. In addition GIFT-Grab exposes a simplified API architecture suitable for video processing pipelines. Wrapping video data as NumPy arrays [2] makes it possible to create video processing pipelines using the rich collection of methods in SciPy [3]. In order to facilitate easy deployment, GIFT-Grab is also bundled as a PyPI package [4]. This allows for a one-liner installation using pip, the recommended tool for Python packages [5] (please see the installation note in the **Availability** section).

GIFT-Grab relies on highly configurable external multimedia libraries and SDKs. Wherever applicable, it uses default configuration options for these. In other words, GIFT-Grab does not attempt to tweak the parameters of these external dependencies unless absolutely necessary. As such GIFT-Grab may not be a suitable choice for applications that have specific requirements, for instance custom video quality.

Appendix A. Full working Python example

The source code listing below shows a full Python example that can be copy-pasted out to a Python interpreter and executed.

```
#!/usr/bin/env python2

from pygiftgrab import IObservableObserver
import scipy.ndimage as ndimage
from pygiftgrab import VideoSourceFactory
from pygiftgrab import ColourSpace
from time import sleep
from pygiftgrab import VideoTargetFactory
from pygiftgrab import Codec

class GaussianSmootherBGRA(IObservableObserver):

    def __init__(self):
        super(GaussianSmootherBGRA, self).
        __init__()

    def update(self, frame):
        data_np = frame.data(True)
        ndimage.gaussian_filter(
                input=data_np, sigma=(5, 15, 0),
                order=0, output=data_np)


if __name__ == "__main__":
    sfac = VideoSourceFactory.get_instance()
    file_reader = sfac.create_file_reader(
        "/tmp/myinput.mp4", ColourSpace.BGRA )

    gauss = GaussianSmootherBGRA()

    tfac = VideoTargetFactory.get_instance()
    frame_rate = file_reader.get_frame_rate()
    file_writer = tfac.create_file_writer(
        Codec.HEVC, "/tmp/myoutput.mp4",
        frame_rate )

    file_reader.attach( gauss )
    gauss.attach( file_writer )

    sleep( 20 )  # operate pipeline for 20 sec

    file_reader.detach( gauss )
    gauss.detach( file_writer )
```

Listing 3: A full Python example that demonstrates a GIFT-Grab pipeline capturing video data from a local file; processing and subsequently encoding processed frames out to another video file. The processing node of this pipeline is the `GaussianSmootherBGRA` class defined in listing 2 and duplicated here for convenience.

Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Tesla K40 GPU used for this research. We would like to thank Dr. Valentina Vitiello for an insightful discussion on the quality management of medical device software.

Competing Interests

The authors have no competing interests to declare.

References

1. **GIFT-Grab on GitHub.** URL: https://github.com/gift-surg/GIFT-Grab [Online; accessed 04-Aug-2017].
2. **van der Walt, S, Colbert, S C** and **Varoquaux, G** 2011 The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science Engineering*, 13(2): 22–30. ISSN 1521-9615. DOI: https://doi.org/10.1109/MCSE.2011.37
3. **Jones, E, Oliphant, T, Peterson, P,** et al. SciPy: Open source scientific tools for Python. URL: http://www.scipy.org/ [Online; accessed 01-Mar-2017].
4. **GIFT-Grab on Python Package Index.** URL: https://pypi.org/project/GIFT-Grab [Online; accessed 04-Aug-2017].
5. **pip.** URL: https://pip.pypa.io [Online; accessed 01-Mar-2017].
6. **FFmpeg.** URL: https://www.ffmpeg.org/ [Online; accessed 25-Nov-2016].
7. **OpenCV (Open Source Computer Vision).** URL: http://opencv.org/ [Online; accessed 16-Mar-2017].
8. **GIFT-Surg Homepage.** URL: http://www.gift-surg.ac.uk [Online; accessed 22-Nov-2016].
9. **Daga, P, Chadebecq, F, Shakir, D I, Garcia-Peraza-Herrera, L C, Tella, M, Dwyer, G, David, A L, Deprest, J, Stoyanov, D, Vercauteren, T** and **Ourselin, S** 2016 Real-time mosaicing of fetoscopic videos using SIFT. Proc SPIE 9786: 97861R–97861R–7. DOI: https://doi.org/10.1117/12.2217172
10. **García-Peraza-Herrera, L C, Li, W, Gruijthuijsen, C, Devreker, A, Attilakos, G, Deprest, J, Vander Poorten, E, Stoyanov, D, Vercauteren, T** and **Ourselin, S** 2017 Real-Time Segmentation of Non-rigid Surgical Tools Based on Deep Learning and Tracking, 84–95. Springer International Publishing, Cham. ISBN 978-3-319-54057-3. DOI: https://doi.org/10.1007/978-3-319-54057-3_8
11. **High efficiency video coding** 2015 Recommendation ITU-T H.265, International Telecommunication Union, Geneva, CH.
12. **Xvid.** URL: https://www.xvid.com/ [Online; accessed 25-Nov-2016].
13. **DVI2PCIe Duo – Epiphan's high performance PCIe capture card.** URL: https://www.epiphan.com/products/dvi2pcie-duo/ [Online; accessed 25-Nov-2016].
14. **Shvets, A, Pavlova, M** and **Frey, G.** Design Patterns Explained Simply. URL: https://sourcemaking.com [Online; accessed 24-Nov-2016].
15. **Support – Epiphan Video.** URL: https://www.epiphan.com/support/ [Online; accessed 25-Nov-2016].
16. **Stroustrup, B** 2013 The C++ Programming Language. *Addison-Wesley Professional*, 4th edition. ISBN 0321563840, 9780321563842.
17. **Sample HEVC Video Files.** URL: https://x265.com/hevc-video-files/ [Online; accessed 4-May-2017].
18. **Information technology – Coding of audio-visual objects – Part 14: MP4 file format**, 2003 Standard ISO/IEC 14496-14:2003, International Organization for Standardization, Geneva, CH.
19. **Dawes, B.** Boost Timer Library. URL: http://www.boost.org/libs/timer/doc/ [Online; accessed 4-May-2017].
20. **NVIDIA Corporation.** NVIDIA System Management Interface. URL: https://developer.nvidia.com/nvidia-system-management-interface [Online; accessed 4-May-2017].
21. **Kitware, Inc. CMake.** URL: http://cmake.org/ [Online; accessed 16-Mar-2017].
22. **Medical device software – Software life cycle processes** 2006 Standard IEC 62304:2006, International Organization for Standardization, Geneva, CH.
23. **Höss, A, Lampe, C, Panse, R, Ackermann, B, Naumann, J,** and **Jäkel, O** 2014 First experiences with the implementation of the European standard EN 62304 on medical device software for the quality assurance of a radiotherapy unit. *Radiation Oncology*, 9(1): 79. ISSN 1748-717X. DOI: https://doi.org/10.1186/1748-717X-9-79.
24. **GitLab.** URL: https://about.gitlab.com/ [Online; accessed 10-May-2017].
25. **GitLab Continuous Integration & Deployment Pipelines.** URL: https://about.gitlab.com/features/gitlab-ci-cd/ [Online; accessed 10-May-2017].
26. **Redmine.** URL: http://www.redmine.org/ [Online; accessed 10-May-2017].
27. **Jenkins.** URL: https://jenkins.io/ [Online; accessed 10-May-2017].
28. **Beck** 2002 Test Driven Development: By Example. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0321146530.
29. **Kitware, Inc. CMake/Testing with CTest.** URL: https://cmake.org/Wiki/CMake/Testing_With_CTest [Online; accessed 16-Mar-2017].
30. **philsquared/Catch: A modern, C++-native, header-only, framework for unittests, TDD and BDD C++ Automated Test Cases in Headers.** URL: https://github.com/philsquared/Catch [Online; accessed 07-Dec-2016].
31. **pytest: helps you write better programs – pytest documentation.** URL: http://doc.pytest.org/ [Online; accessed 07-Dec-2016].
32. **Markdown – GitLab Documentation.** URL: https://docs.gitlab.com/ce/user/markdown.html [Online; accessed 10-May-2017].
33. **Install GIFT-Grab from the Python Packaging Index.** URL: https://github.com/gift-surg/GIFT-Grab/blob/master/doc/pypi.md [Online; accessed 16-Jun-2017].
34. **Blackmagic Design: DeckLink Models.** URL: https://www.blackmagicdesign.com/products/decklink/models [Online; accessed 23-Jan-2017].

35. **NVIDIA VIDEO CODEC SDK|NVIDIA Developer.** URL: https://developer.nvidia.com/nvidia-video-codec-sdk [Online; accessed 25-Nov-2016].
36. **GCC, the GNU Compiler Collection.** URL: https://gcc.gnu.org/ [Online; accessed 25-Nov-2016].
37. **Van Rossum, G** and **Drake, F L** 2003 Python language reference manual. Network Theory.
38. **Abrahams, D** and **Seefeld, S.** Boost.Python. URL: http://www.boost.org/libs/python/doc/ [Online; accessed 16-Mar-2017].
39. **pkg-config.** URL: https://www.freedesktop.org/wiki/Software/pkg-config/ [Online; accessed 07-Dec-2016].
40. **x265 HEVC Encoder/H.265 Video Codec.** URL: http://x265.org/ [Online; accessed 25-Nov-2016].
41. **The WebM Project|Developer Overview.** URL: http://www.webmproject.org/code/ [Online; accessed 07-Dec-2016].
42. **Viitanen, M, Koivula, A, Lemmetti, A, Ylä-Outinen, A, Vanne, J** and **Hämäläinen, T D** 2016 Kvazaar: Open-Source HEVC/H.265 Encoder. In: Proceedings of the 2016 ACM on Multimedia Conference, MM '16, 1179–1182. ACM, New York, NY, USA. ISBN 978-1-4503-3603-1. DOI: https://doi.org/10.1145/2964284.2973796
43. **libVLC.** URL: https://wiki.videolan.org/LibVLC [Online; accessed 07-Dec-2016].
44. **zlib Home Site.** URL: http://www.zlib.net/ [Online; accessed 07-Dec-2016].
45. **Blackmagic Design: Support Center.** URL: https://www.blackmagicdesign.com/support [Online; accessed 23-Jan-2017].
46. **POSIX.1-2008, 2016 Standard IEEE Std 1003.1™-2008, Institute of Electrical and Electronics Engineers, Inc.** and The Open Group, New York, NY, USA and Reading, Berkshire, UK. URL: http://pubs.opengroup.org/onlinepubs/9699919799/ [Online; accessed 07-Dec-2016].
47. **Doxygen: Main Page.** URL: http://www.doxygen.org/ [Online; accessed 25-Nov-2016].
48. **Stoyanov, D, Scarzanella, M V, Pratt, P** and **Yang, G Z** 2010 Real-Time Stereo Reconstruction in Robotically Assisted Minimally Invasive Surgery, 275–282. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-15705-9. DOI: https://doi.org/10.1007/978-3-642-15705-9_34
49. **Pratt, P, Jaeger, A, Hughes-Hallett, A, Mayer, E, Vale, J, Darzi, A, Peters, T** and **Yang, G Z** 2015 Robust ultrasound probe tracking: initial clinical experiences during robot-assisted partial nephrectomy. *International Journal of Computer Assisted Radiology and Surgery*, 10(12): 1905–1913. ISSN 1861-6429. DOI: https://doi.org/10.1007/s11548-015-1279-x
50. **Atasoy, S, Mateus, D, Meining, A, Yang, G Z** and **Navab, N** 2011 Targeted Optical Biopsies for Surveillance Endoscopies, 83–90. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-23626-6. DOI: https://doi.org/10.1007/978-3-642-23626-6_11
51. **Kim, B S, Lee, S H** and **Cho, N I** 2012 Real-time panorama image synthesis by fast camera pose estimation. In: Proceedings of The 2012 Asia Pacific Signal and Information Processing Association Annual Summit and Conference, 1–4.
52. **Leotta, M J** and **Mundy, J L** 2011 Vehicle Surveillance with a Generic, Adaptive, 3D Vehicle Model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(7): 1457–1469. ISSN 0162-8828. DOI: https://doi.org/10.1109/TPAMI.2010.217
53. **Yang, X, Sun, Q** and **Tian, Q** 2003 Content-based video identification: a survey. In: International Conference on Information Technology: Research and Education. Proceedings. ITRE2003., 50–54. DOI: https://doi.org/10.1109/ITRE.2003.1270570

How to cite this article: Shakir, D I, García-Peraza-Herrera, L C, Daga, P, Doel, T, Clarkson, M J, Ourselin, S and Vercauteren, T 2017 GIFT-Grab: Real-time C++ and Python Multi-channel Video Capture, Processing and Encoding API. *Journal of Open Research Software,* 5: 27, DOI: https://doi.org/10.5334/jors.169

Submitted: 16 March 2017 **Accepted:** 24 August 2017 **Published:** 09 October 2017

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press

OPEN ACCESS