A Java-based Framework for Extensible Video Analysis Applications

Dimitris Diamantis, Evaggelos Spyrou, Dimitris K. Iakovidis
Dept Informatics and Computer Technology, Technological Educ. Institute of Lamia
Lamia, Greece
{ddiamantis, vspyrou, diakov}@teilam.gr

Abstract— Information extraction resulting from video analysis is a very important, yet very complex process. In recent years, the growing use of video in video surveillance systems and medical applications has made imperative the existence of user-friendly and cross-platform application frameworks that can be used to extract information from them. However, most of the proposed frameworks are platform-dependent, have a steep learning curve and, unfortunately, force developers to couple their applications with the adopted framework. This makes the resulting applications, platform-dependent, difficult to reuse, hard to maintain and more importantly, coupled with the underlying framework’s application programming interfaces. At an effort to overcome such problems, in this paper we present a novel framework for the development of intelligent video processing and analysis algorithms. The proposed framework, namely Java Video Analysis (JVA) is instantiated by a standard, cross-platform, open access and rich application programming interface. The efficiency, feasibility and performance of the implementation is demonstrated with indicative experimental results.

Keywords—Video analysis, software development framework, medical application

I. INTRODUCTION

The growing amount of multimedia information in the form of digital video is driving the development of content-based, intelligence video analysis and retrieval technologies. However, most of the current methods focus on specific domains, resulting in applications that are hardly generalizable, maintainable and re-usable. Furthermore, the diversity of the available source code implementations makes software integration a rather difficult and resource-demanding process.

To address this issue, we present herein a novel software development framework, with a standard, general purpose, cross-platform and easy to use application programming interface, written entirely in Java, namely Java Video Analysis (JVA) framework. JVA provides an infrastructure for the development of video analysis applications in a standard, reusable and extensible manner. We expect that JVA may motivate further development of standardized open access or even open source video analysis components for many application domains.

State of the art video analysis frameworks have either a methodological or a software development orientation. Frameworks of methodological orientation include an integrated, modular video analysis framework addressing feature extraction and video segmentation [1], a unified framework for object retrieval and mining [2], and a domain-specific framework for colon video analysis[3]. Software development frameworks include computer vision libraries such as the Java Media Framework (JMF), which provides a unified architecture and messaging protocol for managing the acquisition, processing, and delivery of time-based media data [4], OpenCV [5], JavaCV1, and BoofCV [6] and FFMpeg [7], or its Java wrapper Xuggler 2, which address recording, conversion and streaming audio and video.

JVA is a software development framework which provides a formal API under which, different implementations of video acquisition, processing and analysis algorithms (e.g. based on other libraries or frameworks), can be integrated and the code produced is not tightly coupled with the core framework. This maximizes code reusability and system reconfigurability since different implementations can be seamlessly integrated within JVA and accessed in a standard and uniform way.

The rest of this paper is organized as follows: Section II describes the core architecture of the JVA framework, followed by Section which describes the methodology for vision-based capsule endoscope tracking and its implementation with JVA. Experimentation results are presented in Section IV, while Section V summarizes the conclusions of this work.

II. THE JVA FRAMEWORK

In order to gain platform independence, easy integration with the World Wide Web technologies and code reusability while maintain ease of use, JVA framework has been developed using Java. The core framework consists of four independent functional components. Each can be deployed either as a standalone application or as a pluggable entity (plugin).The framework itself provides a standard application programming interface (API) which describes the main functionalities of the plugins, the external reconfigurability of their parameters, I/O and event handling. The plugin-based architecture provides the framework with reconfigurability for the implementation of a variety of video analysis tasks, without any modification to the core component, whereas each plugin can be manipulated independently. Plugin parameters can be configured through an external, extensible text

1 http://code.google.com/p/javacv/
2 http://www.xuggle.com/xuggler
configuration file, or any other key-value pair source (such as a database or even a web service) depending on the implementation. This enables easier experimentation for parameter tuning, and easier access to parameters from external wrapper graphical user interfaces (GUIs) addressing specific applications.

A. Plugin components

JVA consists of four plugin components: Image Data-Source (IDS), Image Processor (IP), Image Analyzer (IA), and Output Handler (OH) component.

IDS enables data acquisition in the form of input image streams. Such streams may originate from a video file with a standard encoding or other sources such as local or remote, cloud image repositories. Three implementations of the Java interface of this component (IImageDataSource) are available: a) FrameGrabber, for image acquisition from standard video files; b) LiveGrabber, for real-time image acquisition from vision sensors; c) StorageDataSource, for image acquisition from a local or remote image repository.

IP enables processing of the input images. After processing, the processed images return to the core framework component. Implementations of the respective Java interface (IImageProcessor) include image cropping and color space transformation, whereas any image processing algorithm, e.g., contrast enhancement, de-blurring etc can be easily incorporated.

IA is responsible for the analysis of single or multiple video frames. Its main functionalities include image sampling, feature extraction, and visualization. The user may define the preferred sampling method policy, use variable sampling rates, implement any image/video feature extraction method, as well as any approach for the visualization of the analysis results through a standard API. This plugin may accommodate supervised or unsupervised machine-learning algorithms, image matching, retrieval and other image/video analysis methodologies. Implementations of the IA Java interface (IImageAnalyzer) include raster scan image sampling, intensity and texture feature extraction approaches and visualization of clustering or classification results on the image space.

Due to the nature and the complexity of IA plugin, feature extraction, Area Policies and Visualization have also been designed as plugins. There are three main components of IA plugin (Fig. 1): Area Policy (AP) is responsible for partitioning images to areas, Feature Extractor (FE) is responsible for feature extraction from given image areas and finally Visualizer (V) handles the visualization of the analysis results.

Those results are then directed to OH, which is responsible for image/video display and storage and also for a possible transfer to remote destinations such as web servers and cloud databases. Three implementations of the Java interface of this component (IOutputHandler) are available: a) the basic implementation of the OH Java interface (IOutputHandler), which is capable of the construction and the local storage of a video file, using visualizations of multiple analyzed input images, b) LivePlay, which may be used to display the output to the screen, c) DiskHandler, which is used to save the output to the local disk. The architecture of the core framework, gives the flexibility to have multiple IOutputHandler plugins at the same time.

JVA Core API (Fig. 2) consists of five packages. More specifically, com.snsp.infrastructure.core contains two interfaces, IConfiguration and IConfigurable. Through IConfiguration interface, custom configuration objects may be implemented, so as to parameterize the framework. The implementation of IConfigurable interface allows objects to consume instances of IConfiguration object; thus it can access the parameters of the whole framework.

In com.snsp.infrastructure.core.events package, IImageDataSourceListener interface is implemented by the core framework in order to listen for events of type IImageDataSourceEvent. Such events are raised from IImageDataSource whenever a new video frame shows up. PluginServiceLocator from package com.snsp.infrastructure.extensibility.common is a singleton which has the responsibility to manage the discovery and the creation of all plugins.

---

This work was partially supported by the Technological Educational Institute of Lamia.
A. Wireless Capsule Endoscopy

Wireless capsule endoscopy (WCE) has revolutionized the field of GI endoscopy [8]. It is performed by a swallowable capsule with the size of a large vitamin that includes a miniature color video camera wirelessly transmitting thousands of video frames during its journey to the anus. It has become a standard imaging technique with a utility that has been extended to the whole gastrointestinal tract, from the esophagus to the colon. It is recommended as a diagnostic or monitoring tool for various diseases including Crohn’s disease, ulcers, polyps and cancer [9].

The tracking of a capsule endoscope within the body is typically performed by external sensors, such as radio-frequency (RF) sensor arrays, which provide only a rough estimation of capsule’s location. We implemented a novel approach to capsule tracking based solely on visual features. Previous works have addressed only the estimation of the rotation angle of the capsule. These include, a method based on Lucas-Kanade optical flow computation (KLT) [10], and a method based on homography estimation [11]. In this paper we extend the latter approach for the localization of the capsule within the GI tract.

The proposed approach is based on a relative pose estimation algorithm proposed by Nister [12]. It involves low-level feature extraction from consecutive video frames, detection of interest point correspondences between them, and estimation and decomposition of the essential matrix.

B. JVA implementation of WCE Tracking

In order to perform WCE tracking, based on our previous work [11], we first extract local invariant features using SURF (Speeded Up Robust Features) [13] methodology between two frames of WCE video. Then, the calculation of the Essential Matrix as proposed by Nister [12], makes us able to estimate camera rotation between those frames based their corresponding points.

The implementation of the described capsule tracking algorithm exploits all plugins described in Section II. In order to extract video frames from WCE video we create the FrameGrabber plugin, which is an implementation of ImageDataSource, utilizing FFmpeg functionalities. Utilizing ImageJ [14] functionalities, we implement ImageProcessor interface, which is used to crop and grey-scale the extracted frames. We crop the each image to create a rectangular sub-image, inscribed within the oval field of endoscopic content. In this study, the 576x576-pixel images are cropped to 450x350 pixels. The cropped and gray-scaled images are provided to IA, for analysis. The implementation of ImageAnalyzer consumes a custom AreaPolicy which describes an initial sampling policy. Initially, according to this policy, a single sample equal to the whole image is provided. Then, this area is sent to an implementation of FeatureExtractor whose role is to extract and describe interest points using the SURF algorithm. At the next step of the algorithm, a custom implementation of Visualizer, uses the results of feature extraction and estimates the essential matrix by using sets of corresponding points within the two images. This provides the flexibility to utilize functionalities of existing...
implementations, such as the one of [6]. The analysis results, which are annotated images (Fig. 3) and distance and rotation estimations, are provided to the custom IOutputHandler implementation. This way we are able to construct a new video, utilizing FFmpeg [7] functionalities.

C. Experiments and Results

The implementation described in the previous paragraph was evaluated with a number of experiments. Those experiments aimed to assess both the feasibility of the capsule tracking methodology and the performance of its Java-based implementation. In Fig. 3 (left) we illustrate a video frame from a typical WCE sequence, along with the positions of the extracted SURF features. We extracted features from 4 octaves and from 4 scales per octave.

RANSAC was then applied on the sets of points of any two given video frames. A set of "tentative" correspondences is created, based solely on visual features. We choose a large number of iterations (i.e. 8000), in order to guarantee that the extracted model will indeed be the best possible. An example of a set of inliers selected by RANSAC is illustrated in Fig. 3.

Since to the best of our knowledge there does not exist a WCE data set with known rotation and scaling factors amongst video frames, we evaluate the proposed method using artificially transformed video frames. The frame of Fig. 3 (left), rotated counter-clockwise by 20° is illustrated in Fig. 3 (right). We provide only the inliers found by RANSAC. For two given pairs of inliers the scaling factor has an estimated value of 1.43 for the case of scaling and of 0.3 for the case of pure rotation. We then select randomly 5 points and apply the relative point algorithm. We get 4 solutions and select the one that makes sense as in [15]. For the examples of Fig. 3, the decomposition of the essential matrix results in an estimated value of 0° for the case of pure scaling and of 21.2° for the case of pure rotation.

In the diagram of Fig. 4, we summarize the processing times of all steps of the proposed algorithm. We have used a small video sequence of 19 sec and extracted frames at a rate of 0.2 sec. This way a set of 100 frames was collected. As it can be observed, the majority of the actual algorithm processing time is dedicated to RANSAC. It was our choice to select a large number of iterations, since we desired the best possible accuracy. However we should note that due to the randomness of RANSAC, the best possible solution may sometimes be found in only a few iterations. Thus, we applied the algorithm 10 times on the same video and their average is the estimate of the processing time of RANSAC.

IV. CONCLUSIONS

In this paper we presented a novel framework for development intelligent video processing and analysis applications using agile methodologies, patterns and best practices, in a standard, cross-platform, open access, rich application programming interface. As an example, a novel methodology for vision-based capsule endoscopy tracking is developed using the presented framework. The use of this framework approach over custom software has the advantage of reconfigurability and extensibility.

REFERENCES

http://java.sun.com/javase/technologies/desktop/media/jmf/


