A SUITE OF TOOLS SUPPORTING DATA STREAMS ANNOTATION AND ITS USE IN EXPERIMENTS WITH HAND GESTURE RECOGNITION

Summary. In this paper we present the concept and our implementation of a suite of tools supporting the annotation of sequential data. These tools are useful in experiments related to multimedia data sequences. We show the two exemplary usage scenarios of these tools in the process of building the gesture recognition system.

Keywords: multimedia annotation, software supporting experiments, point clouds processing, finger alphabet recognition, human-computer interaction

ZESTAW NARZĘDZI WSPOMAGAJĄCYCH ADNOTOWANIE STRUMIENI DANYCH ORAZ JEGO ZASTOSOWANIE W EKSPERYMENTACH DOTYCZĄCYCH ROZPOZNAWANIA GESTÓW DŁONI

Streszczenie. W artykule przedstawiamy koncepcję i naszą implementację zestawu narzędzi wspomagających adnotowanie danych sekwencyjnych. Opracowane narzędzia są użyteczne w eksperymentach związanych z sekwencjami danych multimedialnych. Przedstawiono dwa przykładowe scenariusze użycia tych narzędzi w procesie budowy systemu rozpoznawania gestów wykonywanych dłonią.

Słowa kluczowe: adnotacje multimediów, oprogramowanie wspomagające eksperymenty, przetwarzanie chmur punktów, rozpoznawanie alfabetu palcowego, interakcja człowiek-komputer
1. Introduction

Descriptions of objects and phenomena, data used in experiments, as well as documentation of scientific research are more and more often in the form of multimedia. Usually only certain fragments of collected data are useful. Therefore, the development of tools for annotation of multimedia sequences, which means selecting relevant fragments and describing them, becomes a necessity.

The following, challenging problems need to be addressed:
1. manual (interactive) data annotating;
2. semi-automatic or automatic data annotating;
3. annotating of already acquired and stored data (offline mode);
4. annotating during data acquisition (online mode);
5. development of annotations storage standard;
6. annotating data of any kind (e.g., sequences of point clouds, skeletons or other 3D data acquired from modern imaging devices);
7. automatic referring to existing annotations from the user's program code (their retrieval, insertion, modification, deletion and comparison).

Currently available tools for manual video annotation handle only problems 1, 3, and 5 (see sec. 2). We propose an extension to these tools in the form of software middleware. The first part of it – the framework is helpful in solving problems 2, 3, 4 and 6. The second part of the middleware – the application programming interface (API) for annotation processing is useful in solving the problem 7. Therefore, the developed tools with the program to manual video annotation (e.g. ELAN [26]) form the convenient software environment supporting experimentation with multimedia data sequences.

The remaining part of the paper is organized as follows. Section 2 contains a brief overview of popular tools for video annotation. The general idea of the developed toolset is presented in section 3. Section 4 contains a detailed description of the prepared tools. The exemplary usage scenarios are presented in section 5. Section 6 concludes the paper.

2. Related work

There are several interactive tools for video and audio annotation: ELAN [26], ANVIL [15], VIA [29], Advene [1], VideoAnnEx [27]. They can be used for the manual creation of temporal, multi-layered annotations, attached to entire data chunks (e.g., video frames). In some tools, e.g., ANVIL and VIA, the spatial annotation can be attached to specific areas of video frames or still images [1717][1818]. Another interesting features are 3D viewing of motion capture data
(ANVIL) and real-time annotating during video playback (VIA) [9]. These tools have been used to analyze sign languages [8][30], gestures [7][17], eye movements [5], head pose, velocity and acceleration [21][14], children's touch-screen supported collaboration [10], children's conversational behaviors while interacting with people from cartoon and video [13], music video streams [25], video-based e-learning [2], gesture and speech production for humanoid robots [20], interactions between cognitively impaired older adults and the therapeutic robot [6], and many others.

Manual annotation is a laborious and time consuming task. Several efficient systems of automatic video annotation have been developed [3] [22]. Although there are many solutions, all of them are dedicated to specific areas. Moreover, they are suitable for the processing of audio and video data of common formats and cannot deal with custom, nonstandard streams. Additional problem is accessing the created annotations and corresponding data segments from other tools and scripts.

Therefore, instead of developing another dedicated tool, we propose a universal middleware for annotation systems created in C++ language. It frees the user from the technical aspects of organization and transcription of created annotations, provides the means for accessing and editing them in an automatic manner, and allows for processing and visualization of the custom data formats.

3. General idea

The proposed solution can be used to annotate sequential data, defined as a list of ordered elements (data chunks) referenced by an integer index. By annotating we understand the process of assigning textural information to chosen fragments of this list. We assume that annotations can consist of one or more fields separated by the predefined delimiter character (semicolon by default). The fields allow to describe annotations by additional attributes.

Annotations are organized in co-occurring tiers that can be independent from each other or create a hierarchical structure. Overlapping of annotations is only possible when they belong to different tiers.

The annotations are stored in an additional XML file associated with transcribed media. The chosen file format is compatible with ELAN – one of the most popular tools for manual multimedia annotation.

In addition, a video file composed from 2D views of each data chunk is created. This video and the generated annotations can be loaded to ELAN in order to preview them and make manual adjustments.
A diagram showing general scheme of an annotated data can be seen in Fig. 1. The cubes represent data chunks, the squares – 2D views corresponding to the data chunks, the dark grey blocks – exemplary tiers, and the light grey blocks – exemplary annotations. The three upper tiers are independent from each other and the three lower are hierarchically organized.

The proposed solution has several advantages. The annotation structure is universal and flexible. By the introduction of fields the user can define his own format of annotations, appropriate for a particular use. The original data files remain unaltered. The created annotations can be visualized even for the data that has not direct visual representation.

4. Description of the developed tools

The developed tools consist of a framework to create annotations, API to manage them and the Annotation Wizard – an application with GUI that facilitates the creation of a custom annotation program using the framework.

---

Fig. 1. The general scheme of an annotated data; elements created by the software environment for data annotation are within the area outlined with a dashed line.

Rys. 1. Ogólny schemat adnotowanych danych; elementy utworzone przez środowisko do adnotowania danych znajdują się w obszarze zakreślonym linią przerywaną.
representation (grey rectangles). To create a custom solution, the user prepares classes derived from them. The ways of data acquisition, processing, annotation, visualization and recording are specified by implementation of the virtual methods shown in Table 1.

A single data chunk is represented in the memory as an instance of a class derived from \textit{RawData}. A class derived from \textit{ProcessedData} encapsulates a feature vector, obtained as a result of the single data chunk processing. Both classes have to define the methods \textit{getData} and \textit{getSize}. The first returns a representation of data as an array of bytes, and the second returns the size of this array. Data acquisition from a device (online mode) or a mass storage (offline mode) is performed by a class derived from \textit{DataProvider}. This class should define the methods \textit{getData} returning a single data chunk and \textit{isEndOfData} indicating the end of data stream. The data chunk is processed by the method \textit{processData} of a class derived from \textit{DataProcessor}. A class inherited from \textit{DataAnnotator} is used to determine the annotation start and end points (methods \textit{isAnnotationStart} and \textit{isAnnotationEnd}, respectively) and to classify such extracted fragments (\textit{annotate} method). In the method \textit{get2DDataView} of a class derived from \textit{DataVisualizer}, the user specifies how the single data chunk is transformed to an image. During the runtime the sequence of images, obtained in this way, is displayed. In the online mode data is saved to a mass storage device using a class derived from \textit{DataRecorder}. In the method \textit{appendData} the user defines how to add the single data chunk to a sequence. Moreover, in both program modes, an XML file with annotations and a movie file, composed from the images obtained by the method \textit{get2DDataView}, are created.
Table 1

Abstract base classes with their virtual methods

<table>
<thead>
<tr>
<th>Abstract class</th>
<th>Virtual method</th>
</tr>
</thead>
<tbody>
<tr>
<td>RawData</td>
<td>virtual void* getData() = 0; virtual int getSize() = 0;</td>
</tr>
<tr>
<td>ProcessedData</td>
<td>virtual void* getData() = 0; virtual int getSize() = 0;</td>
</tr>
<tr>
<td>DataProvider</td>
<td>virtual RawData* getData() = 0; virtual bool isEndOfData() = 0;</td>
</tr>
<tr>
<td>DataProcessor</td>
<td>virtual ProcessedData* processData(RawData* source) = 0;</td>
</tr>
<tr>
<td>DataAnnotator</td>
<td>virtual bool isAnnotationStart(ProcessedData* data) = 0; virtual bool isAnnotationEnd(ProcessedData* data) = 0; virtual string classify(vector&lt;ProcessedData*&gt;* segment) = 0;</td>
</tr>
<tr>
<td>DataVisualizer</td>
<td>virtual IplImage* get2DDataView(RawData* data) = 0;</td>
</tr>
<tr>
<td>DataRecorder</td>
<td>virtual void writeRawData(RawData* data) = 0;</td>
</tr>
</tbody>
</table>

The user-defined classes are given to the application by the factory method design pattern [10]. The objects are created during the runtime by creator classes according to the scheme shown in Fig. 3.

4.2. Application programming interface

The developed application programming interface (API) can be used for annotation processing in C++ or Matlab. The annotation file, created by the framework described in section 4.1, is loaded to memory and mapped as a tree of objects that can be easily altered. Modifications include addition, deletion, and edition of single fields, annotations, or whole tiers. The modified structure can be written back to the XML file and visualized by ELAN. There is also the possibility of creating an XML annotation file from the scratch. The query mechanism and the methods for tiers comparison have also been provided. The API consists of the classes shown in Table 2.
Fig. 3. Class diagram showing user-defined class *CloudAnnotator* with associated base and factory class

Rys. 3. Diagram klas przedstawiający zdefiniowaną przez użytkownika klasę *CloudAnnotator* z powiązanymi z nią klasami: bazową i fabrykującą

Table 2

<table>
<thead>
<tr>
<th>Class name</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>CElan</em></td>
<td>Describes the ELAN file structure loaded to the memory, and offers the methods for its processing.</td>
</tr>
<tr>
<td><em>CTier</em></td>
<td>Describes a single tier, that is the sequence of <em>CAnnotation</em> objects, and offers the methods for its processing.</td>
</tr>
<tr>
<td><em>CAnnotation</em></td>
<td>Describes a single annotation and offers the methods for its processing.</td>
</tr>
<tr>
<td><em>CField</em></td>
<td>Describes a single field and offers the methods for its processing.</td>
</tr>
<tr>
<td><em>CElanQueryResult</em></td>
<td>The auxiliary class for storing query results.</td>
</tr>
</tbody>
</table>
4.3. Annotation Wizard

To facilitate the creation of a custom annotation application with the developed framework a graphical user interface called Annotation Wizard is provided. It guides the user through nine consecutive dialog steps to define: raw data (1), processed data (2), data provider (3), data processor (4), data annotator (5), data visualizer (6), data recorder (7), to set parameters (8) and build the application (9). The steps 1 – 7 propose code templates with TODO tags in comments to mark the fragments that should be filled in by the user (Fig. 4). They also allow for third-party software components, needed for completion of a given task, to be indicated in the form of library files (*.h, *.lib, *.dll).

In the step 8, user specifies parameters, such as paths to the compiler and linker, and a video codec used for an AVI file recording. Then in the step 9, command line versions of the Microsoft Visual Studio compiler and linker are used to build the application.

Fig. 4. The step 5 of the Annotation Wizard corresponding to data annotator definition
Rys. 4. Piąty krok kreatora adnotacji odnoszący się do definicji adnotatora danych
5. Exemplary usage scenarios

5.1. Characteristics of the task

The developed tools were used for automatic recognition of a finger alphabet. The finger alphabet is a set of hand gestures corresponding to letters and numbers. It is used by the deaf community for spelling words for which there is no separate sign in the sign language. The gestures are presented with the right hand located at a distance of about 20 cm from the face. Most of them are static (see Fig. 5).

![Fig. 5. Static gestures of the Polish finger alphabet](image)

The hand shapes are complex and the fingers often occlude each other. Therefore, we used the time-of-flight (ToF) camera. It provides the accurate 3D information about the observed scene by measuring the time taken for light to travel from an active illumination source to objects in the field of view and then back to the sensor [28]. The acquired data has the form of a sequence of point clouds. Point cloud is a set of 3D points with additional attributes, e.g., amplitude of the reflected light wave [24].

In the next two subsections we present two exemplary usage scenarios in the process of building the finger alphabet recognition system. In the first scenario the annotating application in the online mode is used while the second scenario present the usage of the offline mode.

5.2. Usage scenario I: Verification of classification results using validation tests

The work plan of the project based on the first scenario is presented in the Fig. 6. It indicates the stages in which the described tools are useful. The left blocks show activities performed with the usage of tools specified in the right blocks (in the tasks in which the tools are used). In this scenario annotating application performs initial segmentation of sequences acquired in real time by a sensor. It adds annotations named "Gesture here" and assigns them to the frames in which a gesture was detected. Then a manual annotation tool is used for
visualization and manual adjustments of annotation ranges. In the last stage the validation tests are performed on the frames specified in the previous step, extracted using the annotation API. If the results are not satisfactory, the whole process is repeated.

Fig. 6. Work plan of the first exemplary scenario of a project with the usage of the proposed annotation tools

Rys. 6. Plan pracy dla pierwszego przykładowego scenariusza projektu z wykorzystaniem proponowanych narzędzi adnotujących

5.3. Usage scenario II: Verification of classification results by visual analysis

The second scenario is presented on the work plan in Fig. 7. In this scenario annotating application performs not only segmentation but also classification of segmented sequences previously recorded in and stored in a file. It generates annotations with names containing classification results and assigns them to the frames in which a gesture was detected. A manual annotation tool is used for visual comparison of results. There is also a possibility
of using the annotation API to automatically compare tiers by generating the additional comparative tier. If the annotation from the tier with obtained results is the same as the annotation from the tier with expected results within the particular range, in the comparative tier the new annotation named "T" is generated within the same range. Otherwise the generated annotation is named "N".

In the later part of the paper we present how to use the developed framework and the API to perform initial segmentation of point cloud sequences. During the acquisition the sequences are split into fragments corresponding to gestures.

![Flowchart](image)

Fig. 7. Work plan of the second exemplary scenario of a project with the usage of the proposed annotation tools

Rys. 7. Plan pracy dla drugiego przykładowego scenariusza projektu z wykorzystaniem propojowanych narzędzi adnotujących

### 5.4. Data representation

In the first step, two classes for raw and processed data encapsulation have been developed.

The class `SR4000RawData`, derived from `RawData` (see Table 2), stores raw data acquired from the ToF MESA Swiss Ranger 4000 camera [19]. Raw data consists of four
arrays $X, Y, Z, A$ of size $176 \times 144$. The first three arrays contain coordinates $x, y, z$ stored as 4-byte floating point values (float type). The array $A$ contains 2-byte unsigned integer values (unsigned int type) corresponding to amplitude of the reflected light wave. A single point of the cloud is then defined as $P = [X(i,j), Y(i,j), Z(i,j)]$, where indices $i, j$ define the same position in arrays $X, Y, Z, A$.

Processed data was encapsulated in the class $\text{InCubeData}$, which is derived from $\text{ProcessedData}$ (see Table 2). It consists of two integer values $N_i$ and $N$ describing a number of points in the cloud within the predefined cuboid area (gesticulation area) for the current and previous cloud, respectively.

### 5.5. Point cloud processing

The point cloud processing was implemented in the method $\text{processData}$ of the class $\text{CloudProcessor}$ derived from the base class $\text{DataProcessor}$ (see Table 2). The place of articulation for the finger alphabet is fixed. Therefore, the vertices of the gesticulation area were set as follows:

\begin{align*}
A1 &= [X(i_f,j_f) + 0.10, Y(i_f,j_f) + 0.15, Z(i_f,j_f) - 0.15] \\
A2 &= [X(i_f,j_f) + 0.40, Y(i_f,j_f) + 0.15, Z(i_f,j_f) - 0.15] \\
A3 &= [X(i_f,j_f) + 0.40, Y(i_f,j_f) - 0.15, Z(i_f,j_f) - 0.15] \\
A4 &= [X(i_f,j_f) + 0.10, Y(i_f,j_f) - 0.15, Z(i_f,j_f) - 0.15] \\
A5 &= [X(i_f,j_f) + 0.10, Y(i_f,j_f) + 0.15, Z(i_f,j_f) + 0.15] \\
A6 &= [X(i_f,j_f) + 0.40, Y(i_f,j_f) + 0.15, Z(i_f,j_f) + 0.15] \\
A7 &= [X(i_f,j_f) + 0.40, Y(i_f,j_f) - 0.15, Z(i_f,j_f) + 0.15] \\
A8 &= [X(i_f,j_f) + 0.10, Y(i_f,j_f) - 0.15, Z(i_f,j_f) + 0.15]
\end{align*}

where $i_f, j_f$ denote indices corresponding to the centre of the face. The face position was determined by the Haar classifier applied on the array $A$ interpreted as an image (see Fig. 8). The implementation available in the OpenCV library [4] was used.

The filters defined in the Point Cloud Library (PCL) [23] were used for the point cloud processing. The pass through filter, which removes points whose selected coordinate is greater than assumed threshold, was applied along $x, y$ and $z$ axis in order to reject all points that do not belong to the gesticulation area. Then, the isolated points of the cloud were removed by the so-called radius outlier removal filter. It rejects points having less than $k$ neighbors. The neighborhood is defined as a sphere of radius $r$. In our case, $r = 0.03m, k = 3$. Then, the number of points was reduced by the so-called voxel grid filter. After applying this filter, the volumetric mesh covering observed scene is created. Then, for each
voxel (a 3D equivalent of a pixel), all the points located within it are approximated with their centroid. In our case the voxel dimensions were $0.01\text{m} \times 0.01\text{m} \times 0.01\text{m}$. The results of the point cloud processing are shown in Fig. 9.

![Fig. 8](image)

**Fig. 8.** Gesticulation area: (a) amplitude image, (b) point cloud front view, (c) point cloud side view

**Rys. 8.** Obszar gestykulacji: a) obraz amplitudowy, b) chmura punktów widziana z przodu, c) chmura punktów widziana z boku

![Fig. 9](image)

**Fig. 9.** Point cloud filtering results: a) original cloud, b) cloud after pass through filter, c) cloud after radius outlier removal filter, d) cloud after voxel grid filter

**Rys. 9.** Rezultaty filtracji chmury punktów: a) oryginalna chmura, b) chmura po zastosowaniu filtru pass through, c) chmura po zastosowaniu filtru radius outlier removal, d) chmura po zastosowaniu filtru voxel grid

### 5.6. Detection of annotation start and end points

The data chunk processed at time $t$ is considered to be the annotation start point if the following condition is satisfied:

$$N_t - N_{t-1} > T_1,$$

where: $N_t, N_{t-1}$ denote number of points in the processed clouds, and $T_1$ is the threshold that, in our case was set experimentally. The condition (2) is checked in the method `isAnnotationStart` of the class `CloudAnnotator` derived from `DataAnnotator` (see Table 2). Similarly, the annotation end point is detected when the following condition is satisfied:

$$N_{t-1} - N_t > T_2,$$

where the threshold $T_2$ is set experimentally. The condition (3) is evaluated in the method `isAnnotationEnd` of the class `CloudAnnotator`. The threshold $T_2$ is slightly lower than $T_1$. This
prevents instantaneous detection of start and end points caused by noise. In our case $T_1 = 145$ and $T_2 = 135$.

For initial segmentation, the method classify of the class CloudAnnotator returns the text "possible gesture (x frames)", where $x$ is the number of clouds belonging to the given annotation.

### 5.7. Data visualization and recording

In the method get2DDataView of the class CloudVisualizer, derived from the class DataVisualizer (see Table 2), an image $I$ of size $176 \times 144$ is created:

$$I(i, j) = \begin{cases} \frac{255(Z(i, j) - Z_{\text{min}})}{Z_{\text{max}} - Z_{\text{min}}} & \text{for } Z(i, j) \in (Z_{\text{max}}, Z_{\text{min}}) \\ 0 & \text{otherwise} \end{cases}$$

where $Z_{\text{min}}$ and $Z_{\text{max}}$ specify the mapped range of $Z$. In our case $Z_{\text{min}} = 0.3\text{m}$ and $Z_{\text{max}} = 1.5\text{m}$. Such visualization resembles a sparse disparity map (or depth map) and is more convenient for the observer than the perspective projection, which is usually used (see Fig. 10).

![Fig. 10. Visualizations of a point cloud: a), b), c) perspective projections - front, side, and top view, d) visualization used by authors](image)

Rys. 10. Wizualizacje chmury punktów: a), b), c) rzut perspektywiczny – widok z przodu, z boku i z góry, d) wizualizacja wykorzystywana przez autorów

In the method appendData of the class CloudRecorder derived from the class DataRecorder (see Table 2), the point cloud obtained from the ToF camera is written to a mass storage device. The XML file with annotations and the video composed from the images used to visualize clouds are created automatically by the framework.

The data fragments corresponding to the created annotations can now be easily accessed from any C++ or Matlab code using the provided API. Moreover the XML and video files can be loaded to ELAN for further manual corrections, e.g., another tier can be created, annotations can be adjusted and labels needed for supervised training of a classifier can be assigned (Fig. 11). The automatically generated tier highly facilitates the process of data labeling, which is important when preparing large training sets.
The developed software presented in this paper have also been successfully used and tested in the project of object tracking based on data acquired from standard color camera. Annotations were generated based on the trajectory of the object's center of mass.

6. Conclusions

The suite of tools supporting the annotation of sequential data has been developed and the usage of these tools in the process of building the finger alphabet recognition system has been shown.

The developed framework and API form a middleware for the researcher preparing his code. They have a universal character and can be applied to a given problem by preparing custom classes derived from the set of abstract base classes. This way the user can focus on the substantive issues (e.g., how to determine the annotations’ start and end points and how to classify the extracted fragment) instead of dealing with low-level technical aspects (e.g., how to organize and store annotations and how to associate them with the corresponding fragments of the sequence). Annotations created with the developed tools can be loaded, visualized and altered in one of the popular programs for manual video annotation. This is
possible even for the custom data format because during the annotation process the associated video file is created. The developed tools can also be used to create data annotation system working in the online mode during data acquisition.

The user is required to have a basic knowledge of object-oriented programming in C++ language, which includes, in particular, understanding of inheritance, abstract classes and virtual methods. The toolkit described in this paper is available at http://vision.kia.prz.edu.pl.

Acknowledgements

This work was partially supported by the NCBR under Grant TANGO1/270034/NCBR/2015.

BIBLIOGRAPHY


Omówienie

W artykule przedstawiamy koncepcję i naszą implementację zestawu narzędzi wspomagających adnotowanie danych sekwencyjnych. Dane tego typu składają się z uporządkowanych elementów (porcji danych) indeksowanych liczbami całkowitymi.

Pierwszym narzędziem wchodzącym w skład omawianego zestawu jest szkielet do budowy aplikacji adnotującej, która może być używana do generowania adnotacji w dwóch trybach: (i) online – w trakcie pobierania danych oraz (ii) offline – dla danych, które zostały wcześniej pobrane i zapisane. Metody wizualizacji danych, segmentacji sekwencji oraz klasyfikacji poszczególnych fragmentów są dostarczane przez użytkownika z wykorzystaniem
programowania zorientowanego obiektowo. Po utworzeniu adnotacji, można uzyskać do nich dostęp, wyszukiwać je i modyfikować przez dowolny program napisany w języku C++ lub Matlab przy użyciu drugiego z omawianych narzędzi – API do przetwarzania adnotacji. Opracowano również aplikację z graficznym interfejsem użytkownika, ułatwiającą tworzenie własnych programów za pomocą szkieletu (pierwszego z omówionych narzędzi).

Opracowane narzędzia razem z aplikacją do manualnej adnotacji danych wideo tworzą wygodne środowisko programowe wspierające eksperymenty z sekwencjami danych multimedialnych. Środowisko to jest uniwersalne i może być stosowane do szerokiej klasy problemów.

Przedstawione zostały dwa przykładowe scenariusze wykorzystania prezentowanych narzędzi w projekcie badawczym, związanym z klasyfikacją obiektów w postaci chmur punktów, w którym wykorzystywane jest proponowane środowisko do adnotowania danych. Na schemacie zaznaczono akcje wykonywane przy użyciu programu do manualnego adnotowania danych, API do przetwarzania adnotacji oraz szkieletu do budowy aplikacji adnotującej.

Pokazujemy również przykład użycia opracowanych narzędzi w procesie budowy systemu rozpoznawania alfabetu palcowego. W opisywanym przykładzie wykorzystujemy dane 3D pobrane przez kamerę time-of-flight. Dla tego typu danych nie istnieją powszechnie akceptowane standardy adnotacji, wizualizacji i przechowywania oraz narzędzia do edycji sekwencji. Omawiany w tym artykule zestaw narzędzi jest szczególnie przydatny w tego typu przypadkach.

Addresses

Tomasz KAPUŚCIŃSKI: Rzeszów University of Technology, Department of Computer and Control Engineering, Faculty of Electrical and Computer Engineering W. Pola 2, 35-959 Rzeszów, Poland, tomekkap@kia.prz.edu.pl.

Dawid WARCHOL: Rzeszów University of Technology, Department of Computer and Control Engineering, Faculty of Electrical and Computer Engineering W. Pola 2, 35-959 Rzeszów, Poland, dawwar@kia.prz.edu.pl.