bio-mathematical modeling, pose recognition,
movement sequences recognition, syntactic description,
natural user interface

Tomasz HACHAJ[1], Marek R. OGIELA[2]

RECOGNITION OF HUMAN BODY POSES AND GESTURE SEQUENCES WITH GESTURE DESCRIPTION LANGUAGE

This paper presents our new proposition of human body poses and gesture description methodology for Natural User Interfaces. Our approach is based on forward chaining inferring schema performed on the set of rules that are defined with formal LALR grammar. The set of rules is called Gesture Description Language (GDL) script while automated reasoning module with heap-like memory is a GDL interpreter. We have also implemented and tested our initial GDL specification and we have obtained very promising early experiments results.

1. INTRODUCTION

The build-in cameras and cheap multimedia devices with USB connectivity became the standard equipment in contemporary home and mobile computer systems. Because of that there is heavy demand on applications that utilizes those sensors. One possible field of application is Natural User Interfaces (NI). The NI is a concept of human-device interaction based on human senses, mostly focused on hearing and vision.

The vision communication with computer program is mainly based on exposing some predefined body poses and movement sequences. Many methods have been yet proposed for extraction and interpretation of those communicates from video stream. In [1] authors propose a method to quickly and accurately predict 3D positions of body joints from a single depth image using no temporal information. The method is based on body part labeling, extracting depth image features and randomized decision forests. In [2] system for estimating location and orientation of a person's head, from depth data acquired by a low quality device is presented. Approach is based on discriminative random regression forests: ensembles of random trees trained by splitting each node so as to simultaneously reduce the entropy of the class labels distribution and the variance of the head position and orientation. Most of the gesture recognition approaches are based on statistical modeling, such as principal component analysis or hidden Markov models [3]. The concept of modeling the dynamic hand gesture using a finite state machine has been proposed in [4]. In [5] a radial basis function network architecture is developed that learns the correlation of facial feature motion patterns and human expressions. Those recognition techniques have many applications not only in games entertainment but also in medicine for example during rehabilitation presses [6].

This paper presents our new proposition of human body poses and gesture description methodology for NI. Our approach is based on forward chaining inferring schema performed on the set of rules that are defined with formal LALR grammar. The set of rules is called Gesture Description Language (GDL) script while automated reasoning module with heap-like memory is a GDL interpreter. In the following paragraph we present in details our approach. We have also implemented and tested our initial GDL specification and we have obtained very promising early experiments results.

[1] Pedagogical University of Krakow, Institute of Computer Science and Computer Methods, 2 Podchorazych Ave, 30-084 Krakow, Poland, email: tomekhachaj@o2.pl.
[2] AGH University of Science and Technology 30 Mickiewicza Ave, 30-059 Krakow, Poland, email: mogiela@agh.edu.pl.

2. MATERIAL AND METHODS

In order to initially test our approach we have created appropriate hardware and software architecture (Fig .1). It is consisted of sensor for data acquisition (Kinect controller), image processing library (OpenNI framework [7]) for user segmentation, skeleton extraction and tracking and reasoning module that is implementation of GDL. That reasoning module is our novel and original contribution.

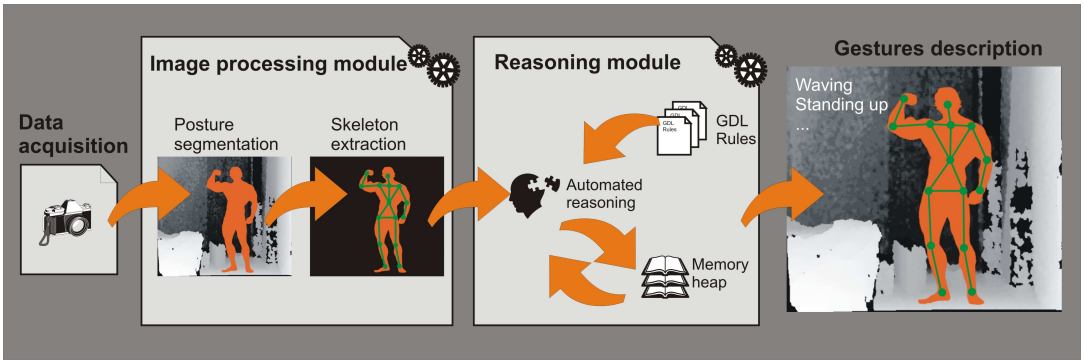

Fig. 1. System architecture. The detailed description is in the text.

The OpenNI framework [7] provides an application programming interface (API) for writing applications utilizing natural interaction. This API covers communication with both low level devices (e.g. vision and audio sensors), as well as high-level middleware solutions (e.g. for visual tracking using computer vision). The API enables modules to be registered in the OpenNI framework and used to produce sensory data. PrimeSense NITE Middleware [8] is a module for OpenNI providing gesture and skeleton tracking. Skeleton tracking functionality enables detection and real-time tracking of fifteen key points on human body (see Table 1 and Figure 2). Those key points will be called skeleton joints (or just joints) in the rest of the article. After processing of depth sensor data NITE returns joint positions and orientations are given in the real world coordinate system. The origin of the system is at the sensor. +X points to the right, +Y points up, and +Z points in the direction of increasing depth. Joint positions are measured in units of mm.

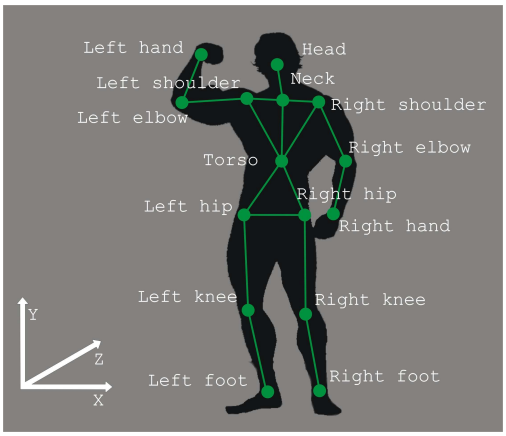

Fig. 2. Skeleton joints and the coordinate system. All body parts are mirrored because user is facing the camera.

GDL language is used for syntactic description human body poses and movement sequences. The GDL script consists of rules set. Each rule has the logical expression and conclusion. If that expression is satisfied the conclusion is added on the top of memory heap. The conclusion from one rule can be present in logical expression of another rule. Multiple rules can have same conclusion. The determination of truthfulness of all rules is made with forward chaining inferring schema by automated reasoning module. On each level of the memory heap automated reasoner keeps information about input data (coordinates of skeleton joints of tracked user) and satisfied rules for given state of memory heap. Each heap level keeps

also timestamp informing (how much time passed since last data addition to the top of the heap). Using this time stamp program can easily check how much time has passed from now to the moment when data was added to the chosen heap level simply by summing up all time stamps from to the top to the chosen heap level.

Table 1. Skeleton joints in OpenNi.

Id	Joint name	Description (if not obvious)
1	Head	Point in the middle of the user's head.
2	Neck	Point between shoulders.
3	Torso	
4	LeftShoulder	
5	LeftElbow	
6	LeftHand	
7	RightShoulder	
8	RightElbow	
9	RightHand	
10	LeftHip	
11	LeftKnee	
12	LeftFoot	Point in the position of left ankle.
13	RightHip	
14	RightKnee	
15	RightFoot	Point in the position of right ankle.

GDL gives an access to heap memory by direct access to actual / previous joint coordinate value (the level of the heap must be specified) or by indirect checking if some of possible conclusions was satisfied in the given time period. All possible elements of GDL are presented and described in Table 2. In GDL scripts letter case does not matter.

The GDL gives direct access to joint data tracked by OpenNI Framework but our algorithm is not limited to this particular software solution or sensor type. The GDL implementation can be easily adapted to any other image processing framework as long as it generates three dimensional joint-based user tracking data.

Table 2. Elements of GDL.

Type	Symbols	Description
Data types		
Number	For example: 3.45, jointName**.x[**heapPosition**]**, result from numeric expression and some functions.	Double precision floating point number. Number might be declared explicitly in the script or is returned as joint coordinate or result from numeric expression and some functions. jointName**.x[**heapPosition**]** returns numeric value of one of joint coordinates (x, y or z). Names of possible joints are presented in Table 1. Numeric value heapPosition determinates position of joint to be retrieved from the memory heap (zero means top of the heap). If joint does not exist in the given position of the heap (for example it was not detected by the image processing library) this expression returns 0.
Point 3D	For example: [3.45,2.11,67], jointName**.xyx[**heapPosition**]**, result from numeric expression and some functions.	Three double precision floating point numbers. Might be declared explicitly in the script or is returned as joint coordinate or result from point 3D expression and some functions. jointName**.xyz[**heapPosition**]** returns numeric value of joint coordinates (x, y and z). Names of possible joints are presented in Table 1. Numeric value heapPosition determinates position of joint to be retrieved from the memory heap (zero means top of the heap). If joint does not exist in the given position of the heap (for example it was not detected by the image processing library) this expression returns [0,0,0].

Logical value	ConclusionName, result from logical expression and some functions.	The binary value (true or false). Logical value is used for checking if the condition of rule in GDL is satisfied. The conclusion of the rule is also a logical value. If conclusion with the given name is present on the top of memory heap it returns logical value true. In not it returns false.
Gesture sequence	**"[**conclusion1,!conclusion2,timeRestriction1] …**[**conclusion3,timeRestriction2**]"**	Sequence of sets of conclusions. Each set of conclusions is in the square brackets. Each conclusion in the set has to be present in the memory heap by the time period specified in field timeRestriction (so the first time period is from current time to current time – timeRestriction1, next period is from current time – time by which all conclusion occurred to current time – time by which all conclusion occurred – timeRestriction2 and so on). If ! precede conclusion that means that given conclusion cannot be present in given time period.
Rule	**RULE** logicalValue **THEN** conclusion	If logical values equals true the conclusion occurs and it is added on the top of the memory heap.
Operators		
Relational operators	**<,<=,>,>=,=,!=**	Binary relational operators between numeric values. If the condition is satisfied it returns logical value true, if not false.
Aritmetic operators	**+,-,*,/,%,^**	Binary arithmetic operators between numeric values (+,-,*,/.%,^) and point 3D values (+,-). "-" can also be an unary operator. ^ returns a specified number raised to the specified power (2^0.5 is a square root of 2).
Logical operators	**&, |**	Logical operators between logical values.
Functions		
Logical functions	**not()**	Negation of logical value.
Numeric functions	**abs(), sqrt()**	Absolute value and square root of numeric value. If the numeric value in sqrt function parameter is negative it generates programming language exception.
Sequence checking functions	**sequenceexists(**"GestureSequence"**)**	Returns true if given "GestureSequence" exists in the memory heap. Returns false if not.
Point 3D functions	**distance(**point3D, point3D**)**	Euclidean distance between two 3D points.
Others		
Brackets	**()**	Bracket can be used for changing the order of operators' execution (arithmetic and logical).
Commentary	//single Line commentary, /* multiline commentary */	

3. EXPERIMENT AND RESULTS

The GDL notation enables description of any body poses and gestures with assumption that gesture can be partitioned into sequences of poses. In order to check the usefulness of proposed semantic description we have created set of scripts that recognize some common behavior that might be present in human – computer interaction. Because of the article space limitation in following text we will show only few examples that represents different capabilities of GDL spirits. The example 1 from appendix shows script that detects movement of the tracked user. The script checks if the position of skeleton joint in torso has changed since last captured frame. The second example from appendix (also figure 3, first row) demonstrates detection of "Psi" pose that is used in various vision systems for calibration purposes. The detection here is only based on description of relative positions between skeleton joints. The third example script detects hand clapping along horizontal axis. The proposed implementation is consisted of two frames: with hands separate (the difference between vertical coordinates of hands has to be under given threshold) and hands close to each other – the difference between vertical and horizontal

coordinates of the hands are under given threshold. The function `sequenceexists` checks if gestures appear in the correct order. The last example (forth example in appendix, also figure 3 middle and bottom row) is detection of hand clapping that is axis invariant. It works in the similar way as script from third example.

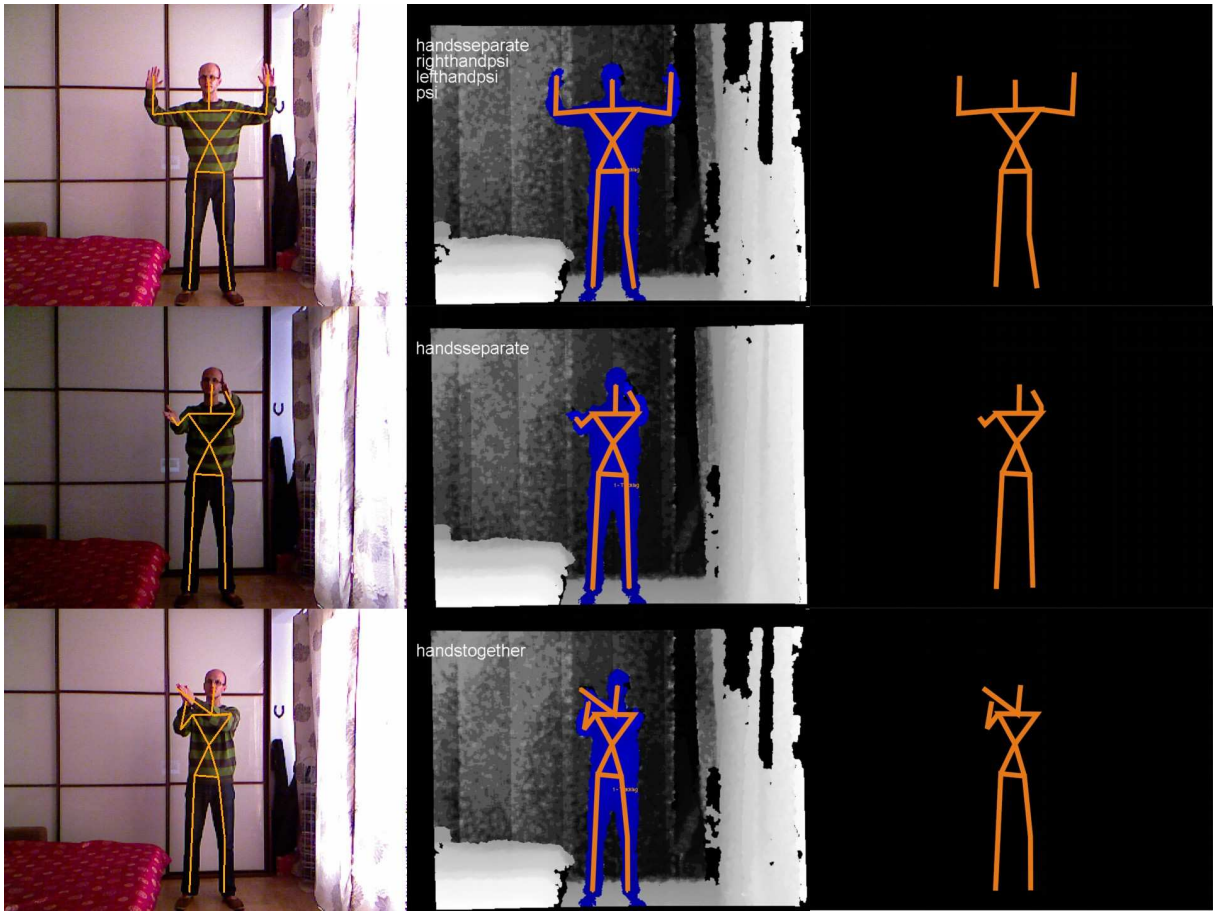

Fig. 3. Two gestures that have been used for description of Psi pose (first row) and hand clapping sequence. In the second row user holding his hands separately, the distance between skeleton joints that represents hands has to be over given threshold. In the last row user holding his hands close to each other – the distance between hands are under given threshold. First column – image from RGB camera, second column – image from depth camera with region of interest (user body) and skeleton detected, third column – tracked skeleton.

Our experiment has proved that GDL scripts processed by our reasoning framework is capable for real time recognition of the considered gestures. That is because any body position can be expressed by inequalities between selected skeleton joints. The `sequenceexists` function restricts the time period in which the sequence should appear. The noises generated by tracking software can be compensated by `abs` and `distance` function. Our early experiments have also showed that some well know gestures (like those we proposed in this section) are easily reproduced (and recognize by our system) by new users who did not have any previous experiences with tracking software.

4. DISCUSSION AND CONCLUSION

It is not a surprise that our semantic description based on formal grammar has yet proven to be reliable in recognition of user behavior represented by few skeleton joints and poses sequences. That is because similar semantic approaches have already been used successfully in many other pattern classification tasks [9].

The main advantages of our approach are simplicity and intuitiveness of GDL scripts. What is more the application of forward chaining inferring schema with memory stack concept is straightforward to any computer programmer and makes development of movement sequences fast and effective. That type of description does not require any training and gathering of huge movement databases. The GDL architect has to decide which skeleton joints can be omitted to simplify the description without affecting

resemblance to exact movement recording. Also GDL does not limit the number and complexity level or rules in the script. Because semantic analysis of once parsed script is not time demanding our approach can potentially has very large rules database.

The main drawback of the methodology is that complex movement sequences might need many key frames. That problem might be solved by finding solution of reverse problem – automatic generation of GDL from filmed movement sequences (similarly to [10]). With proper computed aided tool that would guide the user in process of removing needles joints and setting tolerance level it might be quite effective. From the other hand it should also be remembered that many multimedia vision systems for natural user interfaces use one camera for capturing the movement of user. That fact strongly limits the field of view of the device leading to limitation in observation of some gestures in particular body positions. In those situations some key points of the body might be invisible and difficult to predict by the software. Because of that in many cases one can omit dependences of skeleton joint towards selected axis because the moving sequence will not be properly registered by sensor and tracking software.

The most important goals for future researches are comparison of our approach to other existing methods and validation its sensitivity on test datasets. We are also planning to extend our GDL specification with new functionalities supplying the user with possibility of interacting with virtual object and environment (for example two and three dimensional visualizations of medical images from different modalities [11] or [12]). That would require adding new procedures to GDL and more complex memory stack architecture. We will also consider adding some languages semantic that would allow user to define parts of code that are used multiply times (for example possibility of variables definition).

BIBLIOGRAPHY

[1] SHOTTON F., et al., Real-time human pose recognition in parts from single depth images, CVPR, 2011, 3.

[2] FANELLI G., WEISE T., GALL J., Van GOOL L.V, Real Time Head Pose Estimation from Consumer Depth Cameras, DAGM'11, Proceedings of the 33rd international conference on Pattern recognition, 2011, pp. 101-110.

[3] MITRA S., ACHARYA T., Gesture recognition: A survey, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews , 2007, Vol. 37, No. 3.

[4] YEASIN M., CHAUDHURI S., Visual understanding of dynamic hand gestures, Pattern Recognition, 2000, Vol. 33, pp. 1805–1817.

[5] ROSENBLUM M., YACOOB Y., DAVIS L.S., Human expression recognition from motion using a radial basis function network architecture, IEEE Trans. Neural Netw., 1996, Vol. 7, No. 5, pp. 1121–1138.

[6] OBDRŽÁLEK Š., KURILLO G., HAN J., ABRESCH T., BAJCSY R., Real-Time Human Pose Detection and Tracking for Tele-Rehabilitation in Virtual Reality, Studies in Health Technology and Informatics, 2012, Vol. 173, pp. 320 – 324.

[7] OpenNI framework homepage http://www.openni.org/

[8] Prime Sensor™ NITE 1.3 Algorithms notes, Version 1.0, PrimeSense Inc. 2010, http://pr.cs.cornell.edu/humanactivities/data/NITE.pdf

[9] OGIELA M.R., JAIN L.C (Eds.), Computational Intelligence Paradigms in Advanced Pattern Classification, Springer-Verlag, Berlin Heidelberg, 2012

[10] HACHAJ T., OGIELA M.R, A system for detecting and describing pathological changes using dynamic perfusion computer tomography brain maps, Computers in Biology and Medicine, 2011, Vol. 41, pp. 402-410.

[11] OGIELA M.R., Visualization of perfusion abnormalities with GPU-based volume rendering, Computers & Graphics, 2012, Vol. 36, Issue 3, pp. 163–169.

APPENDIX.- GDL SCRIPTS SOURCE CODES

Example 1. GDL script that detect movement of the tracked user.

```
RULE Distance(torso.xyz[0], torso.xyz[1]) > 10 THEN Movement
```

Example 2. GDL script that detect “Psi pose” of the tracked user.

```
RULE RightElbow.x[0] > Torso.x[0]
& RightHand.x[0] > Torso.x[0]
& RightHand.y[0] > RightElbow.y[0]
& abs(RightHand.x[0] - RightElbow.x[0]) < 50
& abs(RightShoulder.y[0] - RightElbow.y[0]) < 50 THEN RightHandPsi
RULE LeftElbow.x[0] < Torso.x[0]
& LeftHand.x[0] < Torso.x[0]
& LeftHand.y[0] > LeftElbow.y[0]
& abs(LeftHand.x[0] - LeftElbow.x[0]) < 50
& abs(LeftShoulder.y[0] - LeftElbow.y[0]) < 50 THEN LeftHandPsi
RULE RightHandPsi & LeftHandPsi THEN Psi
```

Example 3. GDL script that detect hand clapping along horizontal axis.

```
RULE abs(RightHand.x[0] - LeftHand.x[0]) < 80
& abs(RightHand.y[0] - LeftHand.y[0]) < 80 THEN HandsTogether
RULE abs(RightHand.x[0] - LeftHand.x[0]) > 80
& abs(RightHand.y[0] - LeftHand.y[0]) < 80 THEN HandsSeparate
RULE sequenceexists("[HandsSeparate,0.5][HandsTogether,0.5][HandsSeparate,0.5]") THEN Clapping
```

Example 4. GDL script that detect hand clapping and is axis invariant.

```
RULE distance(RightHand.xyz[0], LeftHand.xyz[0]) < 100 THEN HandsTogether
RULE distance(RightHand.xyz[0], LeftHand.xyz[0]) >= 100 THEN HandsSeparate
RULE sequenceexists("[HandsSeparate,0.5][HandsTogether,0.5][HandsSeparate,0.5]") Then Clapping
```