DE GRUYTER OPEN

Print ISSN: 1312-2622; Online ISSN: 2367-5357
DOI: 10.1515/itc-2017-0009

Fingers and Gesture Recognition with Kinect v2 Sensor

A. Lekova, D. Ryan, R. Davidrajuh

***Key words**: Fingers detection; hand gesture recognition; DTW, contour tracking; Microsoft Kinect V2.*

***Abstract**: The paper presents enhancements and innovative solutions of the proposed in [3] algorithms for fingers tracking and hand gesture recognition based on new defined features describing hand gestures and exploiting new-tracked tip and thumb joints from Kinect v2 sensor. Dynamic Time Warping (DTW) algorithm is used for gestures recognition. We increased its accuracy, scale and rotational invariance by defining new 3D featuring angles describing gestures and used for training a gesture database. 3D positions for fingertips are extracted from depth sensor data and used for calculation of featuring angles between vectors. The provided by Kinect v2 3D positions for thumb, tip and hand joints also participates during the phases of recognition. A comparison with the latest published approach for finger tracking has been performed. The feasibility of the algorithms have been proven by real experiments.*

1. Introduction

In the context of learning new skills by imitation for children with special educational needs in the project [1], we designed a playful environment where a Microsoft Kinect sensor [2] and robotic systems are assistants of the therapist and mediate the process of interfacing objects on digital screens by gestures or navigate fingers of a doll equipped with an artificial robotic hand. The Kinect sensor is a cheap and wildly spread sensor with valuable features, such as a depth sensor and full body joints tracking. However, the Kinect SDK does not support finger tracking. Therefore, we have created algorithms to detect 3D finger positions from depth sensor data. With them we use Dynamic Time Warping (DTW) to recognize gestures by fingers.

A lot of work has been studied for finger recognition by external observations for extracting 3D poses from an image sequence. Typically, Kinect sensor is used for motion-sensing, fingers and gestures recognition [5,6,7]. The common used steps are: (1) depth thresholding; (2) contour extraction; (3) curves detection; (4) fingertips detection; (5) gesture recognition. The methods for gesture recognition could be: DTW algorithms for measuring similarity between two temporal sequences which may vary in speed; a simple rule classifier on sequence of hand poses where gesture is predicted according to the number and content of fingers detected; online fuzzy clustering of the poses over time; simple Hidden Markov Model for classifying of poses and their temporal dependences. The latest one approach for finger detection using the new version v2 of Kinect sensor is presented in [5]. Author detects human hands in the 2D and 3D space defining several thresholds for depth: width, height, max, min, etc., based on DepthFrame and BodyFrame data. He searches for a hand by calculating a distance between tip, thumb and hand coordinates, as well as angles between wrist and hand joints. The algorithm in [5] starts by detecting the Hand joints and the HandStates of a tracked Human body. Thus, the algorithm knows whether it makes sense to search for fingers. The search area is specified by Hand joints positions within a reasonable distance (3D area that is limited between the Hand and the Tip joints, 10-15 cm, approximately). In order to find the contour, any depth values that do not fall between the desired ranges are excluded and thus all depth value that does not belong to a hand is rejected. The outline is the contour of a hand – a big set of points. In order to detect a finger, a "convex hull" in Euclidian space is tried to find. The fingertips are edges of a polyline of the convex hull above the wrist and vertices of convex hull are fingertips if their interior angle is small enough. We experimented the APIs proposed in [5] and found some shortcomings of the algorithm accuracy and speed. Detailed comparison is performed in the last Section. To the best of our knowledge we first use the new-tracked by Kinect v2 tip and thumb hand joints to define the minimum and maximum distances where the hand is located, as it can be seen from interim reports in 2015 for the project referenced in [1].

This study presents innovative solutions and enhancements of the proposed in [3] algorithms for Fingers Detection and Gesture Recognition (FDGR). The APIs for old version of Microsoft Kinect v1 sensor used in [4] have been migrated to Kinect v2. The enhancements in the Dynamic Time Warping algorithm are based on the new-tracked with Kinect v2 tip and thumb joints, as well as rotational and scale invariance of the algorithms achieved by 3D angles rather than 2D positions of fingers as gesture features. Angles, featuring a gesture are calculated based on 3D position extracted by us for each fingertip and provided by Kinect v2 hand and thumb 3D positions. These featuring angles are used for training the gesture database. Thus, we improve the speed and accuracy of algorithms for finger tracking and gesture recognition.

2. Enhancements and Innovative Solutions

The FDGR algorithms, as well as how finger gestures are streaming and recorded with Microsoft Kinect v1 are described in [3]. The APIs [4], presented in *figure 1*, were created with focus on ease of use and the possibility to be customized. The designed by us Kinect-enabled application handles the sensor raw depth data and calculates finger positions and their features over time to recognized gestures using DTW. The identified gesture navigates objects on screens or is used to control the artificial hand motors via Bluetooth connection in real time. With the emerging of Kinect v2 and SDK2 we designed and implemented the enhancements proposed in this Section. Kinect recognizes Human body and populates depth stream of pixels with player index. We do not need the class RangeFinder, since the range is defined in the main class according to BodyFrame data stream for hand joints.

2.1. Contour Tracking

There exists several contour tracking algorithms, such as [8,9,10] however our own algorithm is proposed and implemented. It finds the contour of objects in range of the depth camera. This algorithm works by scanning the depth image from the bottom and up for a valid (belongs to hand) contour pixel. When a valid contour pixel is found it will begin to track the contour of the object that the pixel is a part of. This is done by searching in a local grid pattern with the current found pixel at the origin. The grid extends one pixel in each direction. The search directions are up-left, up-right, down-right and down-left; they are relative to the center of the screen. The starting direction is set to up-left due to the V-shape of the hand above the wrist. The search for a new contour pixel will begin in the same direction as the last pixel was found in. If a new pixel is not found in this direction, searching begins in the next most probable direction, e.g. the most probable direction after up-left is up-right because after searching up along a finger and hitting the beginning of the fingertip the contour direction should change to up-right and down-right. After traversing down to a finger valley a new finger begins, making the next most probable direction up-right. If the algorithm doesn't find a new pixel in the next most probable direction, it starts searching in all the directions, beginning at the last found pixel direction and moving clockwise. The contour tracking algorithm is terminated when the first valid pixel is found or when a fixed number of pixels have been discovered. Details about solving problems of *Single pixel lines* and *Backtracking* algorithm to ensure continuous contour tracking can be found in [3]. The algorithm returns an ordered list with the positions of the contour pixels.

Enhancements: As we explained, to start the contour tracking we need to find a valid contour pixel. A valid contour pixel in [3] is a pixel that is in a specified range from the Kinect sensor. Instead, after migration we use the 3D fingertip coordinates provided by new Kinect v2. The depth detection is performed on Z coordinates of both hand joints. We exploit depth value Z in all vectors containing pixelPosition during the scanning from left, right, traversing horizontally and down. Modified parameters in scanning for the initial pixel are:
MaxPixelsToBacktrack =25;
NumberOfRowsToSkip =2;
MaxEdgePixelCount =700.

The last parameter specifies the maximum number of pixels in order to find all the fingers. We didn't use heightOffset since we use the whole camera space.

2.2. Finger Recognition

Finger recognition consists of three steps. Step one is to find curves on the hand contour. Step two is to find which curves are fingertips and the last step is to find the middle of the fingertip curves. In addition we also get the pointing direction of the fingertips. *Figure 2* shows the results of these algorithms. The red pixels are the extracted hand contour, the yellow pixels are the curve points and the blue pixels indicate where the fingertips are located.

2.2.1. Curve Tracking

The curves detection is implemented using the k-curvature algorithm, proposed in [11]. The k-curvature algorithm detects the angle between two vectors. The implemented version of the algorithm takes three parameters: an ordered list of contour points and two constants k and ω. These constants are application specific, k defines the threshold for contour points taking into account for finding line segments, while ω is an interval of angles (in radians). The algorithm works by creating two vector at each contour point. The first one $\vec{a}$ points to a contour point that is k points forward in the list from the current point. The other vector $\vec{b}$ points to a contour point that is k points backward in the list from the current point. If the contour point list is cyclic, i.e. the contour closes on itself, we create vectors across the start and end boundary of the list. If the list is not cyclic we set $\vec{a}$ to point to the first contour point in the list when the index of the current point is less than k. We must also do the same at the end. We set $\vec{a}$ to point to the last contour point if we are less than k points from the end of the list. We also create a third vector $\vec{c}$ between $\vec{a}$ and $\vec{b}$. The angle θ between $\vec{a}$ and $\vec{b}$. is found according to equation (1). If θ is less than ω we have a curve point.

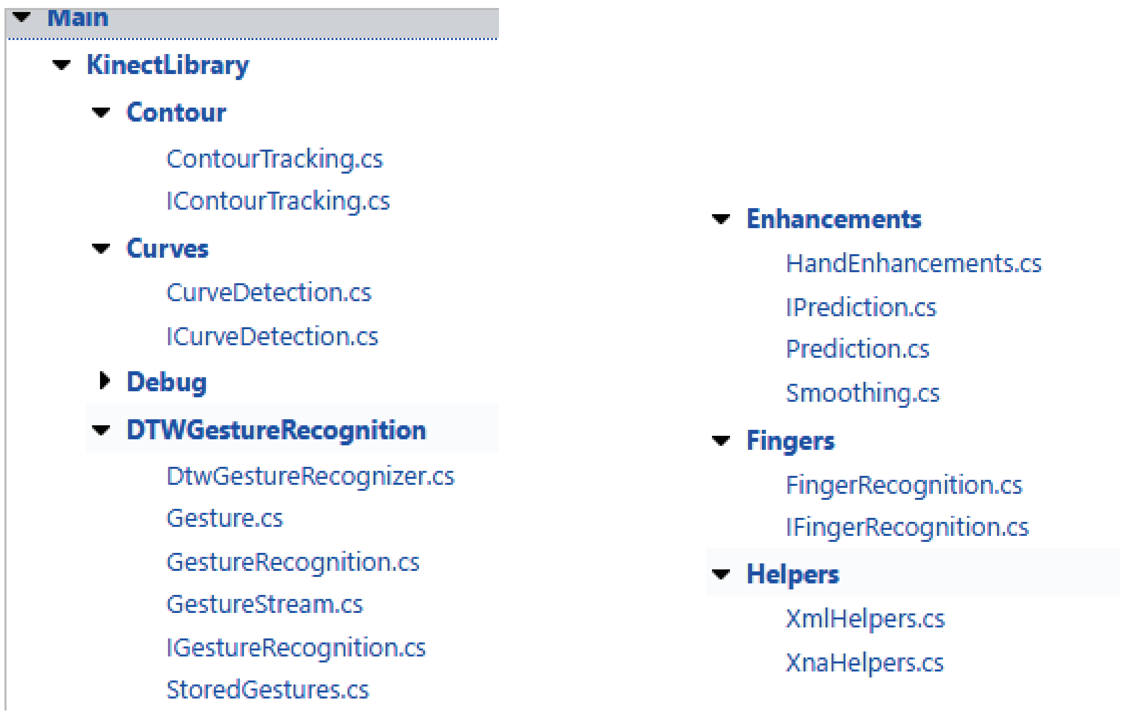
$$(1) \qquad \theta = \cos^{-1} [\vec{a}.\vec{b} \;/\; \|\vec{a}\| \, \|\vec{b}\|]$$

where $\vec{a}.\vec{b}$ is the dot product, while $\|\vec{a}\|$ and $\|\vec{b}\|$ are the lengths.

Enhancements: The k constant specifies how many pixels to travel from the origin point to a new pixel in order to create a vector line segment. This value depends on the application and has been found by trial and error. We established k=20. If k=10 more than five fingers is possible to be found. The min and max angles in the threshold interval for ω (depending on the application) are set to MinAngle = 30 ϵ and MaxAngle=55ϵ.

2.2.2. Fingertips Tracking

We iterate through the curve point list trying to find curve point segments. Curve point segments consists of points that are next to each other. When the start and end points of the curve segment are found, the middle point of the segment is the fingertip location. However, not all seg-



- Main
  - KinectLibrary
    - Contour
      - ContourTracking.cs
      - IContourTracking.cs
    - Curves
      - CurveDetection.cs
      - ICurveDetection.cs
    - Debug
    - DTWGestureRecognition
      - DtwGestureRecognizer.cs
      - Gesture.cs
      - GestureRecognition.cs
      - GestureStream.cs
      - IGestureRecognition.cs
      - StoredGestures.cs
    - Enhancements
      - HandEnhancements.cs
      - IPrediction.cs
      - Prediction.cs
      - Smoothing.cs
    - Fingers
      - FingerRecognition.cs
      - IFingerRecognition.cs
    - Helpers
      - XmlHelpers.cs
      - XnaHelpers.cs

Figure 1. The structure of APIs

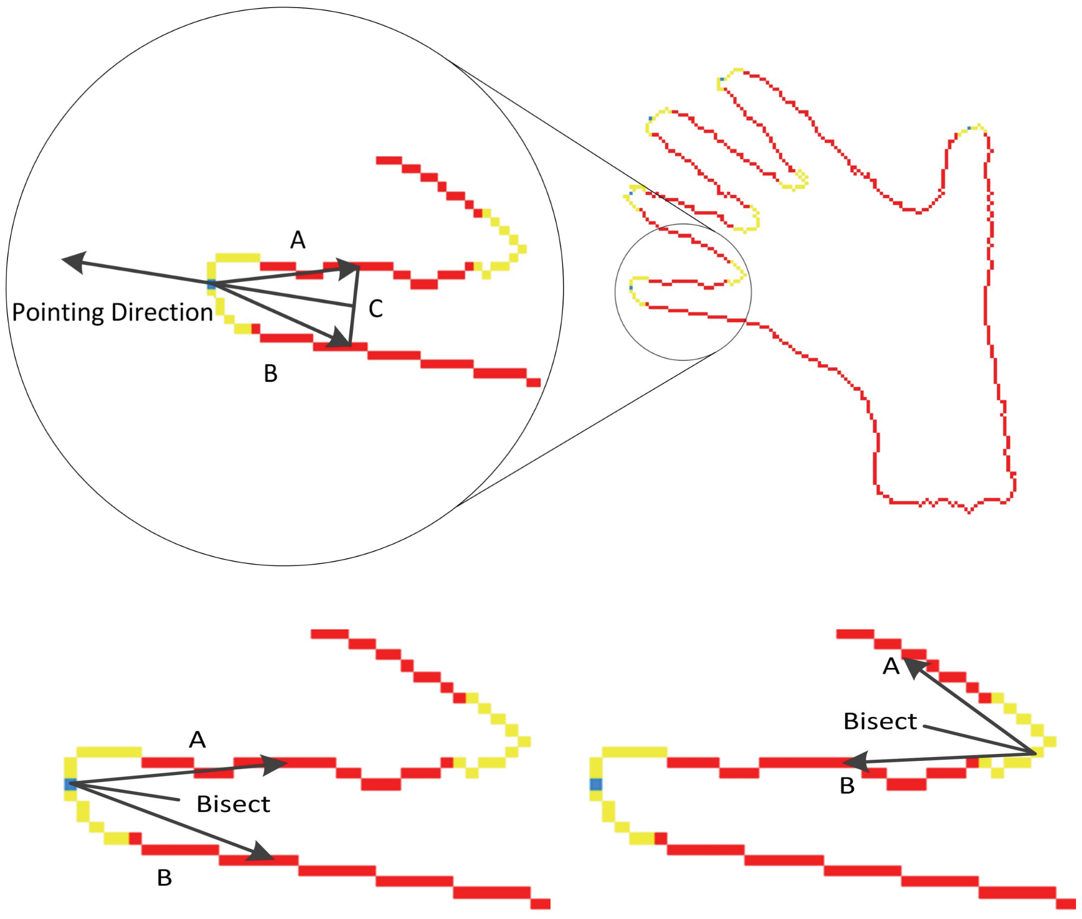
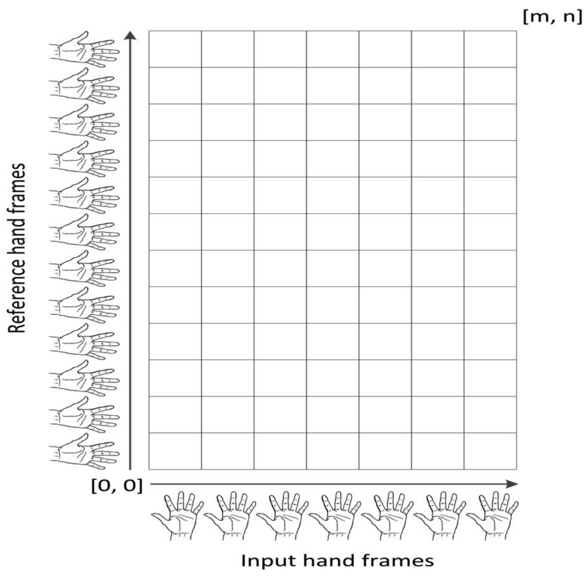
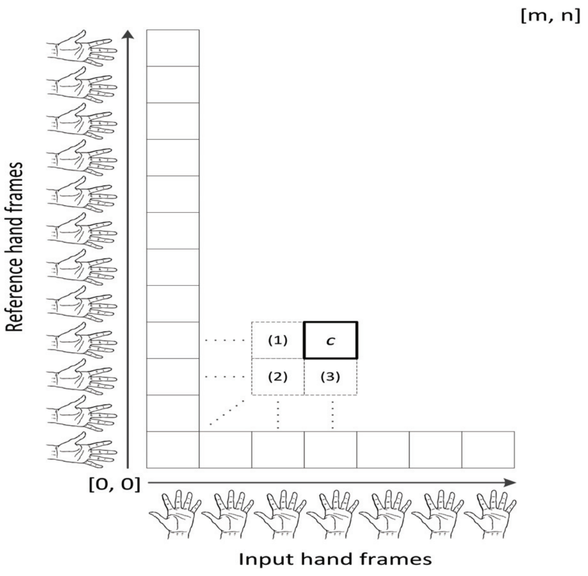


Figure 2. The results of finger detection algorithms



a



b

Figure 3. Hand frame matrix

a) Solving puzzle by gestures

b) Robotic hand for imitation of counting gestures

Figure 4. Real experiments with children

ments are fingertips, they can also be finger valleys. To find if the segment is a fingertip, we create bisect between vectors A and B (see *figure 2*). If the bisect points to a pixel that is in the specified depth range it is a fingertip, otherwise it is a finger valley.

Enhancements of *constants*:
verticalPixelDistanceThreshold=7;
horizontalPixelDistanceThreshold=7.

2.3. Gesture Recognition

A variant of a DTW algorithm is proposed and implemented for gesture recognition. It matches the similarities between two time series according to their features. The two time series do not need to be synchronized in time, enabling a user to do gestures slower or faster than the recorded gesture speed. DTW works in two passes – first a gesture candidate is searched from the last frame in the gesture stream according to its distance cost, calculated by equation (2). Then the accumulated DTW matrix cost is calculated between gesture candidate and recorded gestures in the DB. After finding the candidate gesture in the database, several steps are performed for DTW matrix calculations. First, the cost between each reference and input frames is calculated. The two gestures, a pre-recorded gesture (reference gesture) and the recently performed gesture (input gesture) are compared. This is visualized by using a matrix (see *figure 3*), where *m* is the number of frames for which the gesture stream is recorded, while *n* is the number of frames for the observed (input) stream. Euclidean distance cost *d*(*p,q*) per frame is calculated by equation (2), while total Euclidean distance is calculated by equation (3). The costs between each reference and input frames $c_{cost}[i, j]$ ($0 \le i \le m$ *and* $0 \le j \le n$) are placed in the matrix (see *figure 3a*).

After the matrix is filled with the costs, we compute the lowest accumulated cost matrix. In this matrix we compute the lowest cost to reach a cell. There are three different ways to reach a cell – from the left, bottom or the diagonal down cell. In *figure 3b* is shown that *c* can only be reached by cell 1, 2 and 3. The cost to reach a cell c is the accumulated cost of the one of these three cells added to already calculated $c_{cost}[i, j]$. The lowest of the three calculated

public sealed class Fingertip {public Vector Position { get; set; } public Vector Direction { get; set; } public Vector Bisect { get; set; } }	**public sealed class Feature** { public Vector Angles { get; set; } public Vector K2angle { get; set; } public int frameK { get; set; } }

Figure 5. Hand structures

accumulated cost for *c* gives the final accumulated cost for *c*. In equation (4) c^k is the cost of one of the three cells that can reach *c*. The minimum of the three ones is chosen as the final value for *c*. This step is iterated for all cells, except for cell [0,0], which accumulated cost is set to zero since it is not reachable by any cell.

(2) $d(p,q) = \mathrm{SQRT}((p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2)$

where p and q denotes a finger position in a reference gesture frame and in an input gesture frame.

(3) $d(p,q)_{\_\mathrm{total}} = \mathrm{SUM}(\mathrm{d}(\mathrm{p}_i, \mathrm{q}_i));\ 1 \le i \le 5$

(4) $c_{\mathrm{cost}}\ [i, j] = \min\ (c^k + c_{\mathrm{cost}}\ [i, j]),\ k = 1 \div 3$

During the second phase the lowest accumulated cost path from the last cell ([m,n]) to the first cell ([0,0]) is found. Starting from the last cell, we always choose as the next cell the cheapest one from the left, down and diagonal down cells. The accumulated costs for all cells in the path are summed to give the total path cost. Optimizations and improvements of the steps above for more desirable results can be found in [3].

Enhancements for DTW Gesture Recognizer are based on the new-tracked with Kinect v2 tip and thumb joints, as well as rotational and scale invariance of the algorithms achieved by 3D angles rather than 2D positions of fingers as gesture features. During the both passes in the algorithm we use two type of featuring angles: the first are calculated based on the new-tracked with Kinect v2 3D tip, hand and thumb joint-positions. The second type are featuring angles for fingers, calculated based on 3D position extracted by us for each fingertip and provided by Kinect v2 hand and thumb 3D positions. These features are used for training the database (DB). The format of DB is illustrated in *figure 6*. During the training the gesture stream consists of 42 frames. During the recognition the observed gesture consists of 25 frames. Therefore, each 25th frame initiates new gesture recognition process.

Parameters for thresholds in DTW:
FrameDistanceThreshold=100;
VerticalMovementThreshold =10;
HorizontalMovementThreshold=10;
pathCostThreshold = 50;
maxStoredFrames =42;
maxAccumulatedFrames =25

3. Implementation and Evaluation

3.1. Technical Specification

Kinect v2 sensor is connected to a laptop with Intel(R) Core (TM) i7-5500U CPU@ 2.40 GHz and transfers sensor data to a software application running on the same laptop, built in C++, referencing Microsoft Kinect library (SDK 2) and performing data pre-processing and FDGR algorithms. Kinect SDK middleware could be directly connected to a computer application, such as digital game or two middleware could be connected for remote control. For the present implementation, Kinect SDK2 application on the Kinect side transmits data to Maestro Scripting Language for Maestro USB Servo Controller on the robotic hand side. The code running on the robot side is waiting to receive the data from Kinect side via Bluetooth and use it to make the move of the hand motors.

The enhanced FDGR algorithms have been implemented and tested in two different type of applications: a computer game for solving puzzle by gestures (*figure 4a*) and a robotic hand for imitation of counting gestures (*figure 4b*).

3.2. Data Structures for Hand and Gesture

In each frame, we have processed the depth frame and found the finger positions in the frame. A fingertip is described by a list of vectors Vector(double x, double y, double z). A hand is described by two structures (*figure 5*) – one for 3D positions of fingertips and one for 3D angles. A gesture is a list of hands for each even frame up to 42 (or 25) frames. The gesture database is in XML format, which tags contain hand poses per frame in the gesture stream for different type of gestures. The used gestures are seven, for each gesture we recorded featuring angles for about 8 to 20 examples. The format per frame is given in *figure 6*. Even with the declared above number of training examples, the FDGR works well if finger(s) are moved a little bit in case the gesture is not recognized immediately. More records in the DB improve accuracy, however the time for recognition increases.

3.3. Comparison

We implemented the APIs proposed in [5]. The finger tips are detected in the classes: HandsController and DepthPointEx based on distances and angles between points

```
<Hand>
      <Features>
        <Fingertip3>
          <Angles>
            <X>0.16882305862400515</X>
            <Y>62.1243642290616</Y>
            <Z>769</Z>
          </Angles>
        </Fingertip3>
        <Fingertip3>
          <Angles>
            <X>0.14652809987080839</X>
            <Y>64.292301491964508</Y>
            <Z>759</Z>
          </Angles>
        </Fingertip3>
        <Fingertip3>
          <Angles>
            <X>0.13601450968839465</X>
            <Y>59.357403917683612</Y>
            <Z>762</Z>
          </Angles>
        </Fingertip3>
         <Fingertip3>
          <Angles>
            <X>0.12503074646456028</X>
            <Y>58.679970878013414</Y>
            <Z>764</Z>
          </Angles>
        </Fingertip3>
        <Fingertip3>
          <Angles>
            <X>0.10749157196714702</X>
            <Y>45.298737056613291</Y>
            <Z>780</Z>
          </Angles>
        </Fingertip3>
          <Kinect2angle>
            <X>0.13438510568547066</X>
            <Y>47.985678517288036</Y>
            <Z>814.2622709274292</Z>
          </Kinect2angle>
          <frameK>30</ frameK >
              </Features>
      </Hand>
```

Figure 6. Gesture structure

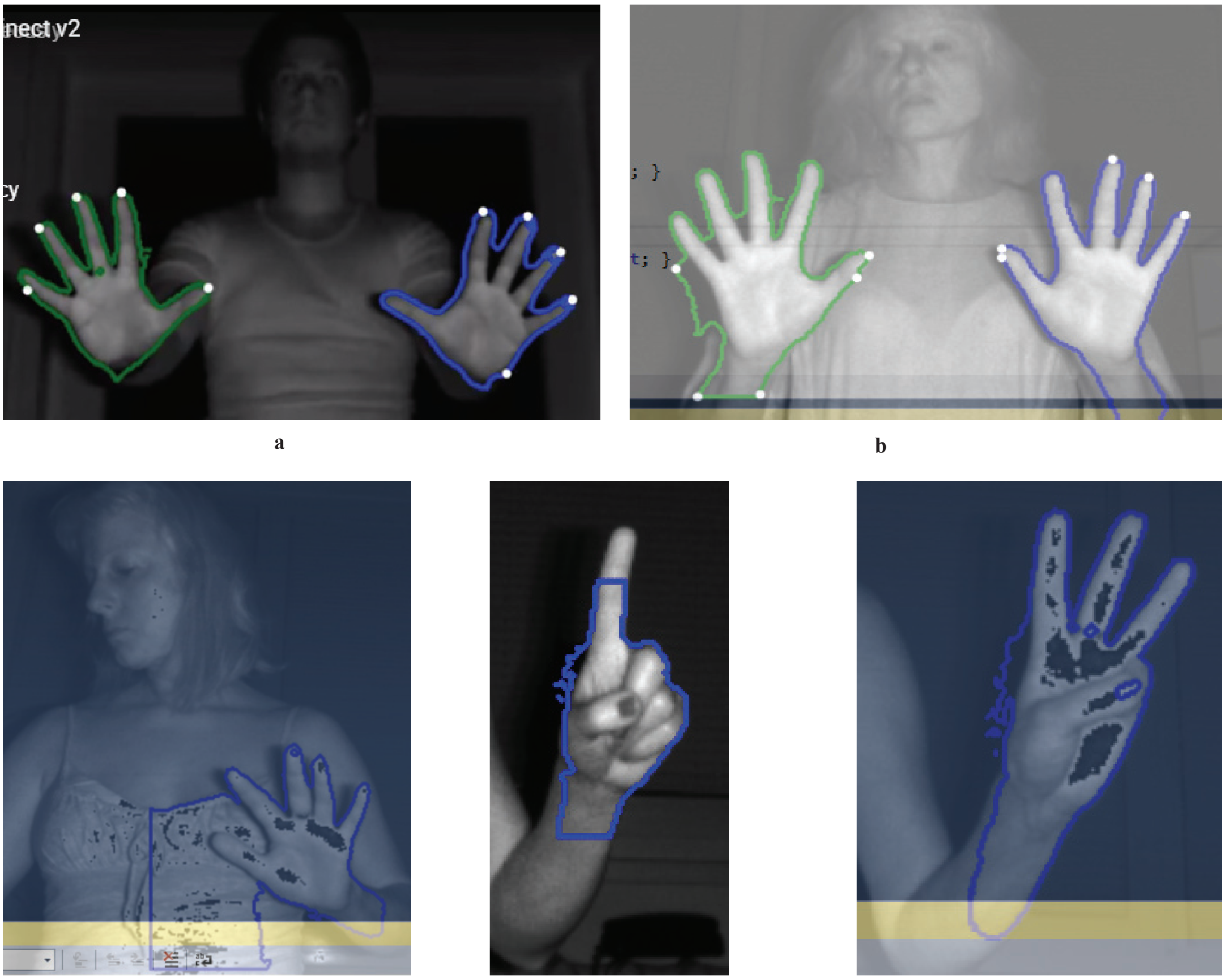
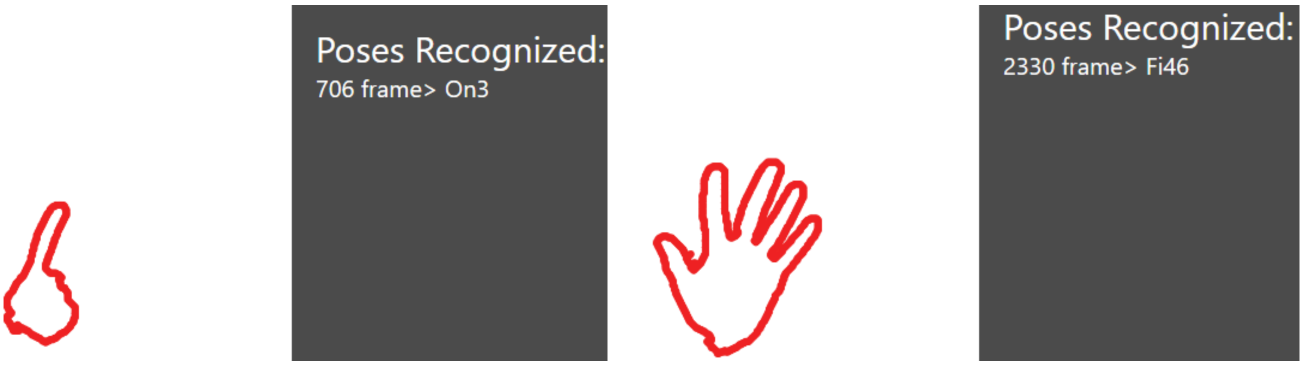


Figure 7. Shortcomings of the algorithm accuracy in [5]



Poses Recognized:
706 frame> On3

Poses Recognized:
2330 frame> Fi46

a) Counting gesture "one" b) counting gesture "five"

Figure 8. Fingertips in gesture recognition

in hand contour. We found out that when the number of fingers in a gesture are one, two or three, the algorithm doesn't detect the point for the fingertip (see *figure 7d and 7e*). When the hand is very close to the body some errors in the contour have been detected, as is shown in *figure 7b and 7c*). The speed of detection per frame could be significantly improved if the video from the camera is not rendered on the screen (camera.Source = frame.ToBitmap()).

In order to prove that the proposed here algorithms are more accurate, we illustrate in *figure 8* the right contour and fingertips detection and recognition of two counting gestures "one" (where only one finger participates) and "five". The more sophisticated algorithms we use for curves and fingertips detection don't penalize the system performance. The feasibility of the proposed algorithms operating in real time has been proven by videos [1], in Section Results> Games for motor and cognitive rehabilitation>Solving_Puzzle and Minion Games. One of the problems we faced was that the hand of the therapist (she very often stands very close to the child) adds more contour points to the hand. In the future, we will provide a solution for filtering the hand of the child based on ID for the first tracked person.

Conclusion

The proposed enhanced and innovative algorithms for fingers detection and gesture recognition have been implemented and tested in two different type of applications. Their feasibility and usability have been proven by real experiments with required accuracy and real time response.

Acknowledgments

This research is supported by the EEA grants, BG09 N D03-90/27.05.2015, Measure "Projects for Interinstitutional Cooperation".

References

1. http://iser.bas.bg/METEMSS/en/.
2. http://www.microsoft.com/en-us/kinectforwindows.
3. Rayan, D. Finger and Gesture Recognition with Microsoft Kinect. https://brage.bibsys.no/xmlui/handle/11250/181783. MSc. Thesis, 2012, 203-208.
4. https://kinectlibrary.codeplex.com/.
5. http://pterneas.com/2016/01/24/kinect-finger-tracking/, 2016.
6. Stein, M. Finger Tracker. 2012. http://makematics.com/code/FingerTracker/.
7. Tang, M. Hand Gesture Recognition Using Microsoft's Kinect. Paper Written for CS229, March 16, 2011.
8. Ren, M., J. Yang, and H. Sun. Tracing Boundary Contours in a Binary Image. – *Image and vision computing*, 20, 2002, No. 2, 125-131.
9. Chang, F., C. Chen, and C. Lu. A Linear-time Component-labelling Algorithm Using Contour Tracing Technique. – *Computer Vision and Image Understanding*, 93, 2004, No. 2, 206-220.
10. Yan, L. and Z. Min. A New Contour Tracing Automaton in Binary Image. IEEE Int. Conf. on Computer Science and Automation Engineering (CSAE), 2011, 2, 2011, 577-581.
11. Trigo, T., S. Pellegrino. An Analysis of Features for Hand-gesture Classification. 17th Int. Conference on Systems, Signals and Image Processing (IWSSIP), 2010, 412-415.

Manuscript received on 02.11.2016

Anna Lekova, *PhD, Assoc. Prof. and Head of the Interactive Robotics and Control Systems Department at Institute of Robotics, Bulgarian Academy of Sciences. She received her MSc in Computer Science (1988) and her PhD in CAD/CAE/CAM from the Technical University – Sofia (1995). Her research interests are in fuzzy-logic for pervasive human-robot interactions, gesture recognition, vision-based motion sensing devices, image processing and pattern recognition, telerobotics. She was coordinator of the EEA Grants project METEMSS (2015-2016) with partners from the South-West University of Blagoevgrad, Bulgaria and University of Stavanger, Norway. In the frame of the project this paper was developed.*

Contacts:
Institute of Robotics, BAS
Bl. 2 Acad. G. Bonchev St., 1113 Sofia, Bulgaria
tel: +359887435648
e-mail: alekova.iser@gmail.com

Daniel Ryan*, receives his master degree in Computer Science from the Department of Electrical and Computer Engineering, University in Stavanger, Norway in 2012. The thesis is entitled "Finger and gesture recognition with Microsoft Kinect". At present, he is currently working as a software developer with focus on design and user experience.*

Contacts:
e-mail: daniel.ryan@outlook.com

Reggie Davidrajuh *has a bachelor study in Physics, a master degree in Control Systems, and a PhD in Industrial Engineering. At present, he is a Professor of Electrical and Computer Engineering at the University of Stavanger, Norway. Dr. Davidrajuh is the editor of the journal International Journal of Business and Systems Research. In addition,he serves on the editorial committees of five other journals. His current research interests are modeling, simulation, and performance analysis of discrete-event systems, algorithms, and graph theory. He is a senior membership of IEEE and a Fellow of British Computer Society. More details about him could be obtained from his homepage: http://www.davidrajuh.net/reggie.*

Contacts:
University of Stavanger
Kjell Arholmsgate 41, Stavanger 4036, Norway
tel: +47 51 83 10 51
e-mail: reggie.davidrajuh@uis.no