FINGERS AND GESTURE RECOGNITION WITH KINECT V2 SENSOR

A. Lekova[1], D. Ryan[2], R. Davidrajuh[2]

[1]*Institute of Systems Engineering and Robotics, Bulgarian Academy of Sciences, Sofia 1113, Acad G. Bonchev str., bl.2, tel. +3599793232, e-mail: alekova@ifi.uio.no*
[2]*University of Stavanger, Norway. e-mail: daniel.ryan@outlook.com; e-mail: reggie.davidrajuh@uis.no*

***Abstract**:* The paper presents enhancements and innovative solutions of the proposed in [3] algorithms for fingers tracking and finger gesture recognition based on new defined features and new-tracked tip and thumb joints with Kinect V2 sensor. Dynamic Time Warping (DTW) algorithm is used for finger gestures recognition. We increased its accuracy and scale and rotational invariance defining new featuring angles calculated from the extracted 3D positions for fingertips and Kinect thumb and tip joints. These features are used for training the gesture DB. A comparison with the latest published approach for finger tracking has been performed. The feasibility of algorithms have been proven by real experiments.

***Key words**:* Fingers detection, Finger Gesture Recognition, DTW, Contour extraction, Curve detection, Microsoft Kinect V2.

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 hand and 3D finger positions from depth sensor data. With them we use Dynamic Time Warping (DTW) to recognize finger gestures.

Existing Solutions. 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 dependence. 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 3D and 2D 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 given 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 range 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 implemented 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 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 as [1].

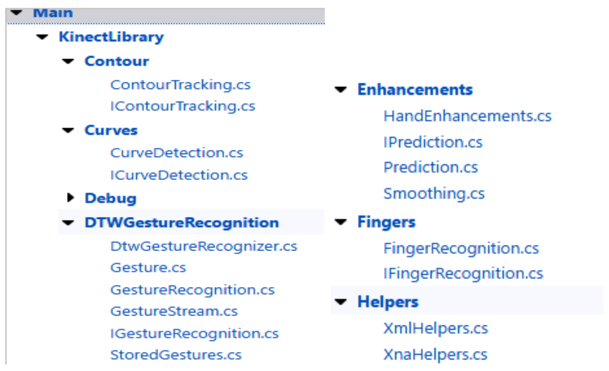
The paper presents enhancements and innovative solutions of the proposed in [3] algorithms for Fingers Detection and finger Gesture Recognition (FDGR). The APIs for old version of Microsoft Kinect V1 sensor used in [4] has been migrated to Kinect V2. The enhancements in the Dynamic Time Warping algorithm 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 for a fingertip, Kinect hand and thumb joint. These features are used for training the gesture DB. Thus, we improve the speed and accuracy of finger tracking and gesture recognition.

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 customize and change core algorithms. 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 below. Kinect recognizes Human body and populates pixel of depth stream 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.

Contour tracking. Own contour tracking algorithm is used in [3]. 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 around the pixel. After the contour tracking algorithm has terminated, the tracking it returns is an ordered list with the positions of the contour pixels.
Enhancements: 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, etc. Modified parameters are: MaxPixelsToBacktrack=25; NumberOfRowsToSkip=2; when scanning for the initial pixel. MaxEdgePixelCount=700; minimum 700 pixels in order to find all the fingers. We didn't use heightOffset because we use the whole camera space.



```
▾ 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 in [3]

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.

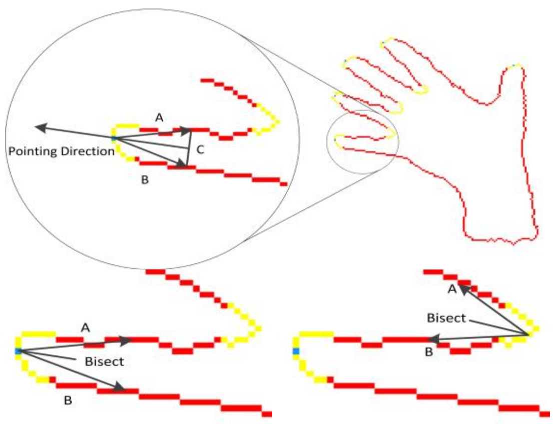


Figure 2. The results of finger detection algorithms

Curve tracking: The curve detection is implemented using the k-curvature algorithm. The k-curvature algorithm detects the angle between two vectors. *Enhancements*: The 'k' constant specifies how many pixels to travel from the origin point to a new pixel in order to create a 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 could be found. The min and max angles in the curvature algorithm also depends on the application. We set MaxAngle=55º and MinAngle = 30 º.

Tracking Fingertips: 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 point of a curve segment is found we find the middle point of the segment. This is the fingertip location. However, not all segments are fingertips, they can also be finger valleys. To find if the segment is a fingertip, we create a bisect between vectors A and B (see Figure 2). If the bisect points to a pixel that is in the specified depth range we know that it must be a fingertip otherwise it is a finger valley. *Enhancements* of *constants*:
verticalPixelDistanceThreshold=7; horizontalPixelDistanceThreshold=7.

Gesture Recognition: To recognize gestures, we implemented a variant of the DTW algorithm, more details about it can be found in [3]. It recognizes similarities between two time series according to its 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 (equation 1). Then the accumulated DTW matrix cost is calculated between gesture candidate and recorded gestures in the DB. DTW matrix calculations consist of several steps after finding the candidate gesture in the database. The two gestures, a pre-recorded gesture (reference gesture) and the newly performed gesture (input gesture) are compared. First, the cost between each reference and input frames is calculated. 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. Calculating Euclidean distance cost *d*(*p,q*) per frame is by equation (1) while calculating total Euclidean distance is by equation (2). The cost between each reference and input frames (c_{cost}) is visualized by using a matrix, see figure 3a. We assume only one hand is used for gestures, although gestures with multiple hands are also possible to be recognized with our solution. 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 3 b. it can be seen that c can only be reached by cell 1, 2 and 3. The accumulated cost for these three cells must be calculated before. If c is the cell we want to calculate and its accumulated cost is c_{ac}, c' in equation (3) is one of the three cells that can reach c. The lowest of the three c_{ac} is chosen as the final value for c.

$$d(p,q) = \mathrm{SQRT}((p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2) \qquad (1)$$

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

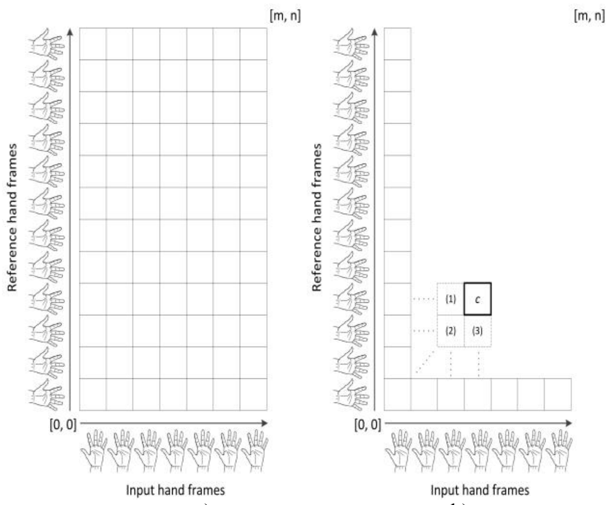
$$d(p,q)_{\_total} = \mathrm{SUM}(d(p_i, q_i)) \;;\; 1 \le i \le 5 \qquad (2)$$

$$c_{ac} = c'_{ac} + c_{cost} \qquad (3)$$

The last step is to find the lowest accumulated cost path from the last cell ([m,n]) to the first cell ([0,0]). Beginning from the last cell, we always choose the cheapest cell of the three we can choose from (left, down and the diagonal down cell) as the next cell. The accumulated cost for each cell in the path is added together to be the total path cost.

DTW Gesture Recognizer enhancements: During the both passes in the algorithm we use two type of feature angles: the first one is calculated based on the new-tracked with Kinect V2 3D tip, hand and thumb joints. The second type is list of featuring angles for fingers, calculated based on 3D position extracted for fingertips, Kinect hand and thumb joints. These features are used for training the DB (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 starts new gesture recognition. *Parameters set:*

FrameDistanceThreshold=100; VerticalMovementThreshold =10; HorizontalMovementThreshold=10; pathCostThreshold = 50; maxStoredFrames =42; maxAccumulatedFrames =25



a) b)

Figure 3. Hand frame matrix

IMPLEMENTATION AND EVALUATION

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: Kinect SDK, on the Kinect side and 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.

a) Solving puzzle by gestures

b) Robotic hand for imitation of counting gestures

Figure 4. Real experiments with children

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

Hand and gesture structure: 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) – for 3D positions of fingertips and 3D angles. A gesture is a list of hands for each even frame up to 42 (or 25) frames. The gesture DB is in XML format, which tags hand poses per frame in the gesture stream for different type of gestures. The used gestures are seven, for each gesture we recorded about 8 to 20 examples. The format per frame could be seen in Figure 6. Even with the declared few training examples the FDGR works well if you move a little bit the finger(s) in case. More records in the DB improve accuracy, however the time for recognition increase.

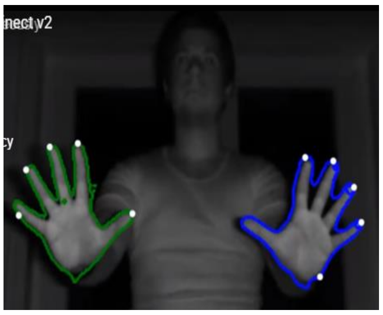
public sealed class Fingertip {	**public sealed class Feature** {

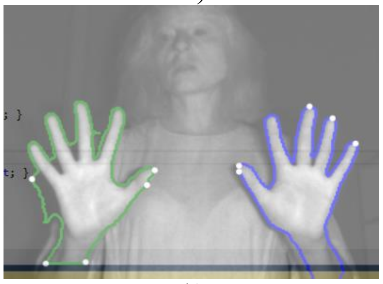
public Vector Position { get; set; }
public Vector Direction { get; set; }
public Vector Bisect { get; set; }
}
	public Vector Angles { get; set; }
	public Vector K2angle { get; set; }
	public int frameK { get; set; }
	}

Figure 5. Hand structures

```
<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>
```

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


a)


b)

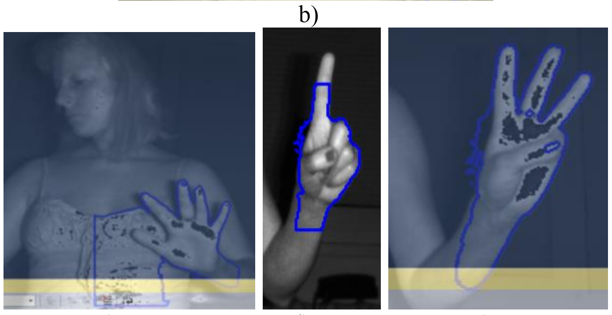
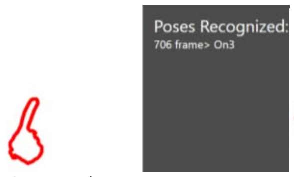
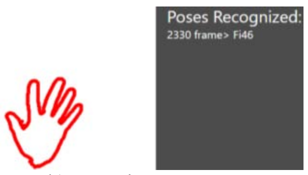

c) d) e)

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

In order to prove that the proposed here algorithms are more accurate, we present on 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 for curves and fingertips detection we use don’t penalize the system performance. The feasibility of the proposed algorithms operating in real time has been proven by videos in [1], Section Results> Games for motor and cognitive rehabilitation>Puzzle and Minion Games. One of the problems we faced was that the hand of the therapist (very often stands close to the child) adds more contour points for 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.


Poses Recognized:
706 frame> On3

a) Counting gestures “one”


Poses Recognized:
2330 frame> Fi46

b) counting gestures “one”

Figure 8.

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.

p.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.