Image processing and gesture recognition in human action-outcome learning experiments

Mangin, Olivier; Toxiri, Stefano; Lonini, Luca

Publication date:
2011

Citation for published version (APA):
Mangin, O., Toxiri, S., & Lonini, L. (2011). *Image processing and gesture recognition in human action-outcome learning experiments*. Paper presented at Capo Caccia Cognitive Neuromorphic Engineering Workshop, Aberystwyth, United Kingdom of Great Britain and Northern Ireland. http://hdl.handle.net/2160/7585

Image processing and gesture recognition in human action-outcome learning experiments

Olivier Mangin, Luca Lonini and Stefano Toxiri

May 20, 2011

Abstract

Microsoft Kinect provides an off-the-shelf sensor that can be used to reliably capture information from body movements in real-time fashion. We implemented an on-line gesture recognition system on top of the kinect's hand tracking capabilities. The system is able to perform real time classification of the user hand gestures by comparing the current movement to a set of 9 predefined template gestures. Gestures are detected when the moving hand exceeds a threshold speed for a minimum duration.

The ultimate goal of this work is to study action-outcome learning in humans: how does a person figure out what actions he can make that have an effect on the environment? How does he shape a gesture to produce this outcome? To this purpose, we improved the recognition algorithm by allowing the dictionary of template gestures to adapt according to the way the user performs the gestures. This allows the emergence of a shared representation of each gesture between the human and the computer, while the user interacts with the system. The approach opens new perspectives in designing and studying interactions between humans and machines as well as in studies of how motor-impaired patients interact with the system.

1 Introduction

In the last years, several electronic devices have been introduced on the market that change the way to interact with machines. Microsoft Kinect is probably amongst the most recent and innovative examples. It allows to control a game console by detecting the movements of the user's body instead of using a hand-held controller. It is based on an RGB camera and two infrared depth sensors, that provide full-body 3D motion capture.

We used the Kinect to detect hand gestures made by the user, in order to explore new paradigms for studying action-outcome learning in humans, i.e. how a user discovers which action will produce an effect on the environment (for example by providing a reward).

We split this problem into two subgoals: as a first step we implemented an algorithm that performs online detection of a gesture made by the user and classifies it amongst a set of 9 template gestures, that were previously stored in the database. In the second step we allowed adaptation of the template gestures according to the user ability to perform each gesture.

Such an approach allows to study how the user and the machine converge to a shared representation of each gesture. The results are discussed in terms of possible experimental paradigms that can be implemented using such a system.

2 The gesture recognition framework

Our setup consists of a PC running MATLAB (The Matworks, Natick, MA) under UNIX and a Microsoft Kinect connected via USB. We used two set of libraries, OpenNI (`http://www.openni.org`) and NITE (`http://www.primesense.com`) to interface the Kinect with Matlab and acquire the position (x, y) of the hand on the screen at a sample rate of 30 Hz. The capture window resolution is 640x480. All the software runs under Matlab.

The recognition system is based on the following elements:

- a segmentation algorithm that extracts gesture candidates from the incoming hand movement stream,
- a normalization step, that computes a reference representation of the gesture invariant to translation, rotation, and scaling,
- a distance measure between sequences of hand positions, that allows comparison of normalized gesture data to a reference dataset (in the simplest case).

2.1 Segmentation into gestures

The first thing the recognition system must handle is to decompose incoming hand movements into gestures. A simple criterion is to require the hand to be still at the beginning and end of a single gesture.

The segmentation algorithm we implemented is based on the following elements. A hand movement is detected as the beginning of a gesture when the following two conditions are met:

$$C_1 = \begin{cases} v > v_{th,on} \\ t > T_{min,on} \end{cases}$$

where v is the hand speed (i.e. the norm of the velocity vector) and t is the time since v exceeded the threshold $v_{th,on}$.

The end of a gesture is defined by the following conditions:

$$C_2 = \begin{cases} v < v_{th,off} \\ t > T_{min,off} \end{cases} \quad \text{OR} \quad t > T_{lim}$$

where T_{lim} is the maximum duration allowed for a single gesture.

Such values are set as $v_{th,on} = 90px/s$; $T_{min,on} = 1s$; $v_{th,off} = 30px/s$; $T_{min,off} = 0.1s$. Different thresholds are used to enhance robustness of the system to noise. Hand speed is computed by averaging the derivative of the 3 last positions samples.

The code implemented for recognition of a gesture is based on a state machine made up of 3 states: *Wait*, *Record* and *Compare* (Figure 1). While in the *wait* state, the position of the hand is acquired but no data is saved. When condition C_1 (Gesture Detection) is verified, hand position is saved until condition C_2 is verified. Afterwards the system switches to the *compare* state, where the saved gesture is compared, as detailed in the following, and then returns to the *wait* state. The code runs in real-time, i.e. it does not require any offline computation for recognition.

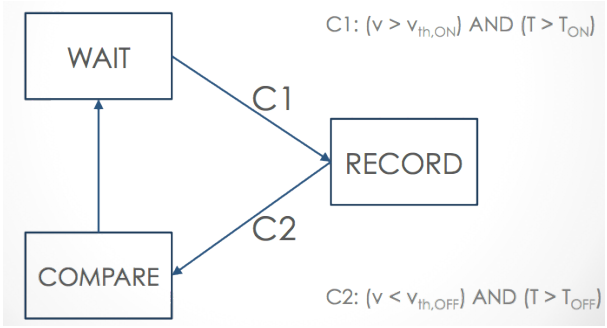

Figure 1: The state machine that implements the gesture recognition.

2.2 Normalization

Once a gesture has been acquired, the sequence of positions $P = ((x_1, y_1), ..., (x_N, y_N))$ is transformed to create a representation invariant to translations, rotations and scaling of the gesture.

The result of such transformations is shown in Figure 2.

Translation invariance Invariance to translations is obtained by centering the sequence of positions with respect to the starting position (x_1, y_1), i.e.

$$(x_i, y_i)_C = (x_i - x_1, y_i - y_1),\ \forall i \tag{1}$$

By using this transformation and assuming that the resulting sequence of positions is a *standard* representation of the gesture, we have introduced a new convention: the gestures must always begin at the same point. For very symmetric gestures such as geometric shapes (circles, square, triangles) the symmetries of the shape relax this constraint so that every vertex of the shape is a possible start (actually for squares and triangle the following step is required for that), and starting from vertices is a natural choice for a demonstrator so this constraint is not really sensible from a user point of view.

Rotation invariance The shape is rotated about its start velocity vector v_s so that this is parallel to the x-axis. More precisely, since all the gestures begin with zero velocity, we average the first 3 samples of the velocity that exceeds a fixed threshold (i.e. $v_{th} = 5$ px/s).

The rotation is given by the following equations:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix}_{CR} = R \cdot \begin{bmatrix} x_i \\ y_i \end{bmatrix}_N,\ \forall i \tag{2}$$

where R is the rotation matrix

$$R = \frac{1}{|v|^2} \cdot \begin{bmatrix} v_{s,x} & -v_{s,y} \\ v_{s,y} & v_{s,x} \end{bmatrix} \tag{3}$$

We bring to the reader's attention the fact that this transformation yields invariance to rotations but not to path direction. For example, a circle drawn clockwise or counter-clockwise will not lead to the same representation. A simple way to address this issue is to duplicate the incoming gesture into two gestures: the original one and the symmetric with respect to the x-axis. Each one is submitted to the recognition system and the best result is then kept.

The symmetric gesture is computed by multiplying each datapoint by the matrix S with:

$$S = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{4}$$

Scaling invariance Finally, data is scaled by the inverse of a measure of its general width. We use the trace norm of the data matrix, which is the square root of the trace of the empirical covariance matrix of the sequence of positions.

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix}_{CRS} = \frac{1}{\sqrt{Tr(Cov(P))}} \cdot \begin{bmatrix} x_i \\ y_i \end{bmatrix}_N,\ \forall i. \tag{5}$$

The previous sequence of transformations provides an invariant representation, that proved to be quite efficient in our first experiments.

It however has some strong limitations coming, first of all, from the assumptions that grounded this transformation (gestures always start from the same point, scaling and rotations are irrelevant, orientation of starting velocity, etc.). Another limitation comes from the lack of robustness of these transformations. For example, it is clear from Figure 2 that a small difference in the starting velocity vector for the triangles will yield to quite different sequences of points. This limitation could be managed by changing the width of the window used to compute the average start velocity.

On the other hand, this representation is dependent on the chosen convention for the gestures. Alternative assumptions may be:

- to use average or end position instead of starting position,
- to compute the direction from starting point to center of mass, or the direction of the eigenvector of the covariance matrix associated to the largest eigenvalue,
- to use any other norm on the covariance matrix or simply the largest distance between two points, to do the scaling.

Another approach would be to try and build local features of the movement that are invariant by the required transformation. For example Frenet framing allows to represent acceleration in such a way. Many features could be used simultaneously in a higher dimensional vector, and the sequence of such vectors could replace the original sequence.

Finally other pre-processing could be applied to the hand positions data, such as low-pass filtering or re-sampling, in order to reduce noise in the data.

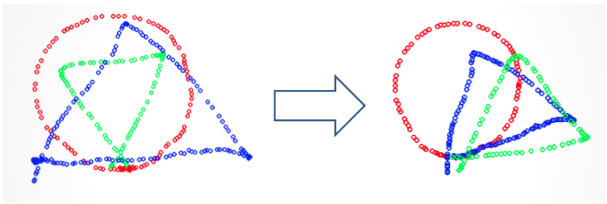

Figure 2: Three gestures acquired from the system before (left) and after (right) normalization.

2.3 Classification of the gestures

We use a simple nearest neighbor classification algorithm, by comparing the candidate gesture to each example of a dataset and keeping the label of the best matching.

Since we are dealing with sequences of vectors of different length, it is neither useful nor possible to use simple Euclidean distance to compare gestures. We thus use dynamic time warping (DTW) distance [Sakoe and Chiba(1978)], a variant of the edition distance, which is able to automatically detect the best alignment over the sequences of vectors and compute an overall distance from vector to vector Euclidean distances. The alignment procedure allows flexibility in the matching of vectors, thus compensating for the length difference and other time deformations.

We used DTW Matlab implementation from [Ellis(2005)].

3 Adaptation to the user

In our first experiments with the recognition system, we noticed that the users were sharply adapting their gestures to what the system seemed to expect.

In order to allow further studies and make some simple modeling of this behavior, we implemented an adaptation mechanism in the system.

3.1 Gesture representation

We introduce here a new internal representation of gestures, which is more compact and easy to manipulate than the raw sequence of movements. The idea is to start from the information we already have, that is to compare the current gesture to the templates of a dataset.

Until now we were comparing a given gesture g to each template $t_i \in \mathcal{T}$ where $\mathcal{T} = \{t_i | 1 \leq i \leq n\}$ is our template set and n is the number of templates. This operation yields a vector $d(g) = (d_{DTW}(g, t_i))_{1 \leq i \leq n}$ from which we are only using the minimum. d_{DTW} is the Dynamic Time Warping distance introduced in Section 2.3.

We will now use the whole distance vector to compute a gesture representation. In the previous case, taking the minimum is equivalent to transform a gesture into a vector of scores which has all zeros except at the position corresponding to the best matching template where we then have a one. In other words, a dimension of the space is associated to each template and each gesture is associated to the axis vector corresponding to the closest template.

We compute a *score-like* vector based on a relaxation of this mechanism, by using an analogous of the **softmax** function (actually here it is more a softmin). The score vector is given by the following

formula:

$$s(g)_i = \frac{\exp\left(-\frac{d(g)_i}{\sigma}\right)}{\sum_{j=1}^{n} \exp\left(-\frac{d(g)_j}{\sigma}\right)} \tag{6}$$

where σ is a parameter of the function.

We first notice that when $\sigma \longrightarrow 0$ we tend to go back to the previous behaviour.

This representation projects each vector on the first quartile of the n-dimensional sphere. The σ parameter sets how close we are to the axis (small σ) or to the first diagonal (large σ).

3.2 Comparing and updating gestures

Given a gesture g, its score s(g) represents a vector into a new space, where gestures are compared. This vector can be seen as an internal representation of the gesture in the system. We call the transformed template gestures as **prototypical gestures**. We thus see both incoming gestures and reference gestures as vectors of this space.

The two operations we want to do on such vectors are: comparing gestures (i.e. an incoming gesture with a prototype) and updating internal representation of gestures (i.e. update a prototype to better fit an incoming gesture).

Gesture comparison We use scalar product as a similarity measure between gestures (sometimes this is also called *cosine distance*). More precisely, to compare two gestures represented by v_1 and v_2, we compute the following:

$$v_1 \cdot v_2 \tag{7}$$

Internal representations update Prototypical gestures v are moved closer to an example gesture g performed by the user through the following:

$$v \longleftarrow \frac{(1-\eta)\cdot v + \eta \cdot s(g)}{\|(1-\eta)\cdot v + \eta \cdot s(g)\|} \tag{8}$$

where η is an update parameter (i.e. learning rate).

3.3 Experimental setup

We built the following experimental setup, in order to demonstrate the capabilities of this system. A user stands in front of the system, and is given hints about the kind of gestures the system is supposed to recognize (for example the name of a shape). Then the user performs gestures and tries to get them recognized by the system. As a feedback he is prompted with the name of the recognized gesture as well as a recognition score (plus a green or red background depending on whether the recognition threshold was exceeded or not).

The system uses a template dataset to compute the previously described representation and have internal representations of the gestures corresponding to the templates, initialized as the basis vectors of the representation space. Then when a gesture is thought to be recognized by the system (after being compared to all prototypes, the most similar has a similarity above the threshold), the corresponding representation is updated using equation (8).

For visualization purpose, if two or three templates are used, the internal representation space can be easily plotted. Such visualization is shown in figures 3.

Another important detail is the fact that templates used to compute the gesture representation and internal models of gestures need not to correspond to the same gestures and need not to have same cardinal. Actually the computation of the compact gesture representation and the comparison to internal gesture prototypes that evolve are two completely separated process. The number of template will only impact the richness and precision of the internal representation. It is thus possible to learn completely new gestures directly in those representation, without updating the template dataset.

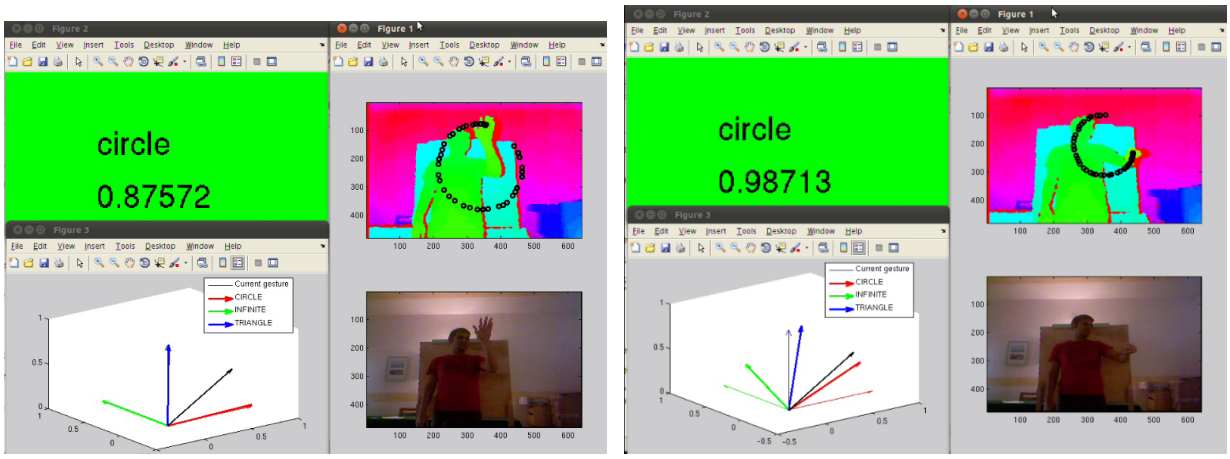

Figure 3: 3a Recognition with the adaptive algorithm for trial 1: (Top Left) The user's gesture is recognized as 'Circle' and the score is displayed below the name; (Bottom Left) The 3 prototype templates ('Circle', 'Triangle' and 'Infinite') as well as the current gesture are displayed into the feature space; (Top Right) The depth map produced by the Kinect sensors. The black trace corresponds to the last 20 hand points detected; (Bottom Right) view from the Kinect RGB camera.
3b Recognition with the adaptive algorithm after 20 trials. As it can be seen the score for the recognized 'Circle' is increased (0.875 to 0.987) and the prototypes have moved. Note that prototype of 'Circle' is now closer to the user's gesture that corresponds to a circle.

4 Discussion

Development of low cost sensing technologies is changing our way to interact with machines. From traditional keyboard or keypad controllers, we are shifting towards an era were machines can detect our own movements without holding any device.
By using Microsoft Kinect, we implemented a gesture recognition system that is able to reliably classify a specific gesture of the user by tracking of the user's hand. The only required constraint is that the each gesture starts and ends with a velocity that is close to 0.
Furthermore, by representing gestures into the space of prototype gestures (i.e. normalized distances from templates), it is possible to adapt the prototypes to match the user's way of performing a specific gesture. What emerges is that both the user and the machine adapt themselves to converge into a shared representation of the gestures.

This simple setup opens new and exciting possibilities for the study of action-outcome learning. In fact, the system can be used to study how people adapt their movements to discover a rewarding action. This in turn connects to an highly active topic in neuroscience, that is the biological mechanisms that regulate discovery of novel actions [Redgrave and Gurney(2006)]. Moreover, studies on motor impaired patients, such as Parkinson patients, can be run with such a setup to quantitatively evaluate their ability to perform and adapt to specific gestures.

References

[Ellis(2005)] Daniel P. W. Ellis. Dynamic time warping (DTW) in Matlab, 2005. URL `http://www.ee.columbia.edu/~dpwe/resources/matlab/dtw/`. Online web resource.

[Redgrave and Gurney(2006)] P. Redgrave and K. Gurney. The short-latency dopamine signal: a role in discovering novel actions? *Nature Reviews Neuroscience*, 7(12):967–975, 2006.

[Sakoe and Chiba(1978)] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1): 43–49, 1978. ISSN 0096-3518.