(This is a sample cover image for this issue. The actual cover is not yet available at this time.)

J. Vis. Commun. Image R. 23 (2012) 426–440

Contents lists available at SciVerse ScienceDirect

J. Vis. Commun. Image R.

journal homepage: www.elsevier.com/locate/jvci

Playing into the wild: A gesture-based interface for gaming in public spaces

Marco Roccetti *, Gustavo Marfia, Angelo Semeraro

University of Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italy

ARTICLE INFO

Article history:
Received 1 July 2011
Accepted 28 December 2011
Available online 17 January 2012

Keywords:
Computer games
Image processing
Immersive environments
Action recognition algorithms
Gesture following
Gaming in public spaces
Tortellini pasta
Shanghai 2010 World Expo

ABSTRACT

Gestural-based interfaces have become one of the fundamental technologies that can determine the success of new computer games. In fact, computer games today offer interaction paradigms that go well beyond the use of remote controls, letting players directly perform exchanges with the objects and characters that compose the virtual worlds that are displayed in front of them. To perform such exchanges, new algorithms and technologies have been devised which include advanced visual recognition schemes, new video cameras and accelerometer sensors. At the same time, other important trends are also quietly emerging in the same domain: game designers, in fact, are slowly shifting their attention out of the walls of gaming fanatics homes, broadening their interests to computer games that can be played in public spaces, as exhibitions and museums. However, to the best of our knowledge, only a very limited amount of research experiences have taken into account the problem of producing computer games, based on gesture-based interfaces that well suit such settings. Hence, in this paper we address the problem of differentiating the design of a gesture-based interface for a console from the problem of designing it for a public space setting. Moreover, we will show that within a public space, it is possible to narrow down the vision algorithms that can well support the recognition of complex actions, whereas solely relying on a simple webcam. In particular, we will describe the design and implementation of an interface that well suits public immersive scenarios, since it is based on a simple and efficient set of algorithms which, combined with the intelligence given by the knowledge of the context of where a game is played, leads to a fast and robust interpretation of hand gestures. To witness this last aspect, we will report on the results obtained from the deployment of a computer game we specifically developed for public spaces, termed Tortellino X-Perience, which has been enjoyed by hundreds of visitors at the 2010 Shanghai World Expo.

1. Introduction

Beyond any doubt, the major producers of computer games are engaged in a technological race that aims at providing their respective customers with the most exciting and entertaining human-computer interface. Recent trends show that games that adopt true-to-life interfaces have higher chances of beating their competitors in the market share arena. This means that customers are no longer happy of traditional interfaces (e.g., joysticks, keyboards, mice, etc.), but desire to play computer games that let them perform body movements that are close to those that would be performed in reality. For example, when hitting a ball, players prefer swinging their arm or leg, rather than sliding a mouse or pressing a button.

We can trace back the beginning of such process to the introduction of the Nintendo Wii, console that first supported gesture-based interactions with games using a combination of action recognition and motion tracking algorithms. Although still bound to the use of a controller (i.e., the Wiimote), the Nintendo Wii has succeeded in letting players move freely, as playing on a track or a field. Now, a further step forward has been made with the creation of the Microsoft Kinect interface, composed of an advanced video camera and a software system that jointly support the recognition of body movements for the Xbox console. In Kinect-supported games, players can move naturally and contemporarily control their avatar, which, mimicking their movements, implements a direct correspondence between any movement performed in reality and the equivalent movement performed in *virtuality*. However, this has required the use of *range cameras*, hardware devices that, combined with infrared sensors, are capable of estimating the distance from the objects that are captured in real-time, thus adding 3D information (i.e., depth) to 2D images. Although the cost of such type of equipment is slowly decreasing, now falling in the price range of a hundred dollars, it is still worth searching for alternative solutions that, implemented with simple video cameras, can heavily cut the budget required for the deployment of multiple gaming systems. In fact, many of such alternative solutions exist, given the extensive amount of research that has been devoted to the detection and tracking of physical objects in computer vision. However, as we

* Corresponding author.
E-mail address: roccetti@cs.unibo.it (M. Roccetti).

1047-3203/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jvcir.2011.12.006

shall explain in Section 2, these solutions are typically resource intensive and can affect the overall performance of a computer game's execution, given that they already dedicate important amounts of computation and memory power to support complex graphical scenarios.

As gesture-based interfaces rapidly mature, their demand steadily increases, constantly broadening their range of applications. In fact, these interfaces are no more confined within the controlled boundaries of private homes, being also employed in public settings, as for example museums and exhibitions. Interestingly, in public, new problems arise, as interface designers are required to respond to a completely different set of constraints. Players, for example, are not limited to the inhabitants of houses, and the area reserved for gaming is typically much larger than that delimited by a living room. Also lighting conditions, in general, could radically differ from those found within private homes. Summarizing, the assumptions that were true when designing gesture-based interfaces for the home console market segment may now be no longer so within public settings.

The main contribution of this work is, in terms of vision researches, to show that a set of algorithms exists that, opportunely customized, meet the requirements that public spaces impose. We realized a set of algorithms that are specifically designed for gesture-based interfaces for public spaces. These algorithms are composed by simple hardware and software components, and are aware of the context where any interactions take place. In fact, differently from other experiences, they do not require the installation of any specialized hardware other than a computer, with normal computational capabilities, and a webcam, thus not requiring the implementation of any high resource-consuming algorithm for the tracking of player actions. However, unlike other solutions, our software components benefit from the awareness of the context where they operate, as the context where the game takes place is not, as within home settings, an external factor out of the control of the game designer, but part, itself, of the interface. Now, within this scenario and with the design aim of finding a robust solution that may work efficiently for any type of player, we will show that it is possible to narrow down the algorithms that can well support the recognition of complex actions to very simple ones.

Anticipating now how the algorithms work, a hand segmentation scheme analyzes the real-time video stream captured by a video camera, which is placed above the area where a player will move his or her hands, and identifies as valid motion patterns all those movements that are detected at certain key spots. These key spots are well illuminated and positioned to be relevant to the given game, so that all those events that would not have any effect on the game can be pruned in advance. In cascade, an innovative hands recognition algorithm simply and effectively determines the positions of the player's hands, found as the far ahead positions where motion is detected. Finally, actions are recognized checking for their timing and for the correctness of their trajectories by an action recognition algorithm. If the actions a player performs are correct, he or she can appreciate their effects in the virtual world of the game. For example, when playing with a virtual ball, the action recognition algorithm ensures that the ball displayed on the screen changes its trajectory only when this is hit at the right time and in the right place.

Summing up, the advantages given by our algorithms, specifically designed for public settings, are manifold: (a) they are flexible and easily adaptable to new settings, given that that they are completely implemented in software, not needing any specialized hardware components, (b) compared to others that also solely use a simple video camera but are only limited to motion tracking operations, ours also detects the player's hands, while they wave within the gaming area, (c) they support the real-time tracking of player movements and gestures, and (d) they are not locked to any existing console or proprietary hardware platform. All such features broaden the use of games to new segments of customers, supported by a variety of different hardware technologies.

As a final consideration, we demonstrated in practice that our approach particularly suits exhibition gaming scenarios. To witness this fact, we developed a computer game, termed Tortellino X-Perience, which has been shown at the Shanghai World Expo in 2010, at the Cantina Bentivoglio and at the International Tortellino Constest held in Bologna in 2011 [1–4,21–25]. All these events represented important opportunities for checking the validity and robustness of our approach through a widespread assessment campaign performed thanks to all the players that tested our game (i.e., a couple of hundreds in a few days). The objective of the game was that of teaching players how to prepare the Tortellini Pasta starting from its main ingredients (flour, water and eggs), while challenging them through each step of its recipe. Clearly, the preparation of a Tortellino requires walking through a variety of phases and the final outcome heavily depends on how well the cook approaches each of them. For this reason our gesture-based interface played a very important role, being able to distinguish in real-time correct from incorrect movements.

The rest of this paper is organized as follows. In Section 2 we provide an overview of the main approaches that have been, to this date, chosen in the design of gesture-based interfaces. In Section 3, instead, we describe the main challenges involved in the design of algorithms that support such type of interfaces for public spaces. Section 4 provides an overview of the ideas that lie behind the design of our algorithms; Section 5 describes our test-bed and its performances, whereas Section 6 deals with extensions that could broaden the applicability of our algorithms. We conclude with Section 7. A final Appendix provides the implementation details of our algorithms.

2. Background and related work

In the past decade, a wealth of research in academia and industry has focused on finding new and more natural means of interaction between humans and computers. Many of such initiatives aimed at devising new algorithms and technologies for home gaming scenarios, devoting little, or no attention to the development of interaction schemes for immersive public settings. For this reason, we will here describe a set of technologies that have not been deliberately devised to be used within museum or exhibition scenarios, but have either been integrated within popular home gaming consoles or represent promising developments of computer vision techniques that, to this date, have not found their way in commercial products [5–15].

The most widespread controller that has been to this date created, capable of supporting the tracking of a player's movements is the Nintendo Wiimote controller [16,17]. The Wiimote is equipped with an infrared camera sensor, which provides high speed tracking of up to four IR light sources. This infrared camera computes the distance of the Wiimote from two light emitting sources placed within the sensor bar, which is typically positioned above or below the TV set. The Wiimote then transmits such information in real-time to the console via a Bluetooth interface. Moreover, such controller also carries an accelerometer, used to register any sudden acceleration that is experienced by a player's hand while engaged in a game. Clearly, using such technology requires that players hold a Wiimote, hence demands the use of a hardware broker between the gestures a player performs and a Wii console. This hardly suites a public scenario, where hundreds or even thousands of people can join and play a game subsequently using the same console during a single day, as such hardware devices are prone to get broken or lost when intensely utilized by

many non-expert players. Very recently, computer game players have been able to enjoy body free gaming with the Xbox, thanks to the introduction of the Microsoft Kinect sensor. Kinect, in brief, is a horizontal bar that, as the Wii sensor bar, is placed either above or below the video screen. However, differently from the Wii sensor bar, it adds the capabilities of a depth sensor to those of an RGB camera, hence the ability of recording the distance from all objects that lie in front of it. The depth information is then processed by a software engine that extracts, in real-time, the human body features of players, thus enabling the interaction between the physical world and the virtual one [18]. Sony, before Microsoft, has offered its customers mixed reality experiences with the EyeToy first and with the Playstation Eye more recently. Both of these products are based on a digital camera that feeds the captured video stream to software algorithms, designed to infer game input commands. Compared to our approach, the mechanisms employed within each of the abovementioned popular home gaming systems present at least one of the following problems, when brought within the context of a public space: (a) the use of hardware devices, which can jeopardize the realistic experience of hands-free human-computer interactions (e.g., Nintendo Wii), and requires the use of proprietary devices, with additional costs (e.g., Microsoft Kinect), and (b) the need of a calibration phase, an idle time interval before any new player begins his/her game (e.g., Microsoft Kinect, Sony EyeToy and Playstation Eye).

From the theoretical point of view, a number of methodologies have been devised for tracking an object that moves in real-time. Two of the most promising ones, are based on the use of the Hough Transform and of Particle Filters, respectively [19,20]. The Hough Transform, whose initial scope was that of identifying the presence of simple contours (e.g., lines, circles, ellipses, etc.) in figures, has later been extended to detect more complex patterns. To understand how this methodology works, let us consider the problem of discovering whether a set of points on a plane all belong to the same line. With the Hough Transform, each point that lies on the 2D Cartesian space (x_0,y_0) is mapped onto a sinusoid in the Hough space (r,θ), given by the parametric representation of a line given by equation $r = x_0 cos\,\theta + y_0 sin\,\theta$. Now, if a set of points in the 2D Cartesian space is collinear, the corresponding sinusoids in the Hough space intersect all at the same point, thus providing a simple way for checking their collinearity. Later, this methodology has been extended to support the detection of arbitrary shapes, as for example the shape of a hand within the frames that compose a video stream. Particle Filters, instead, are an extension of Kalman filters, whose purpose is to estimate the value of a variable from a noisy measurement. While Kalman Filters operate on the assumption that an observation, for example z_t at time t, is the result of the sum of the desired signal, say x_t, and Gaussian noise (e.g., n_t), Particle Filters are more general, as they can be used to analyze observations z_t that result from nonlinear distortions of the desired signal added to non-Gaussian noise samples (e.g., $z_t = f(x_t) + k_t$, where $f(\,)$ is a nonlinear function and k_t is the result of a noisy non-Gaussian stochastic process). This is performed by building a posterior probability $P(x_t|z_{0:t})$, which in Kalman filters is assumed to be Gaussian, as the sum of weighted samples or particles. The more observations are made, more samples are available and more accurate is the estimate of the posterior probability $P(x_t|z_{0:t})$, which hence provides a more efficient filtering operation. Although both the Hough Transform and Particle Filters represent very interesting mathematical tools for the detection and tracking of objects that move within a video stream, in practice, both are limited by excessive processing power and memory space requirements to be effective in real time. In fact, the computational burden of the Hough Transform exponentially increases with the number of model parameters (i.e., the number of parameters required to identify a given shape). Particle Filters, similarly, show an excessive increase of particles, and hence of computational power requirements, as the model dimension increases. For this reason, a number of research initiatives have been investigating the opportunity of implementing both of such techniques utilizing parallel hardware architectures, but paying a cost, in terms of flexibility. Concluding, to this date it appears unfeasible to employ such type of techniques in public gaming scenarios, as the disadvantages given by the amount of resources they require exceed the advantages given by their performances.

3. Home vs. public scenarios

To set the stage to the description of the new challenges and opportunities that lie ahead of computer game designers, we should first move one step backwards and provide a brief historical introduction. Since their birth, computer games have spread within homes, but also through public spaces, where arcade games were often found in businesses such as restaurants, cafeterias and pubs, for example. Arcade games remained popular, for long, until they lost much of their attractiveness in the mid-nineties when 3D-powered video consoles stepped in. Now, almost two decades later, new technical innovations (e.g., gestural interfaces) are opening the gates to new paths, which include the design and implementation of computer games that can be played anywhere and anytime, while not being confined to the walls of a private home or to the hardware of a remote control interface.

This said, the scope of this section is twofold. In the first part we will analyze what differentiates designing action recognition algorithms for public settings, while, in the second, we will walk through the steps that led us to the devise of the vision algorithms that implement a gestural interface that well suits public scenarios.

3.1. Design issues

We can generally find one main difference between a game that is played at home, from a game that is played in a public scenario: the location. This simple fact changes the approach that a game designer should keep, when moving between the two. In a public scenario, in fact, the context where the game is played is fully accessible to any type of player (including the game designer), while this is clearly not true at home. Moreover, in a public scenario, anyone can step in and play, while this cannot typically happen at home. This means that the *context accessibility* and the *player heterogeneity* are two variables that change when moving between the two scenarios. What is important to understand, and will emerge in the following, is that accounting for these two variables is key for a successful design of a computer game, especially when using gestural interfaces.

With context, in particular, we identify both the location and the characteristics (e.g., accessibility, available space, illumination level, etc.) of the location where a game is played. The availability of an accessible game location implies that public space game designers can, in principle, fully control the context where their games will be displayed, contrarily to home game designers that, instead, have no clue regarding where their systems will be finally deployed, with obvious consequences on the flexibility that is required from their systems. To overcome this problem, home game consoles and human-computer interaction sensors are typically shipped with all sorts of recommendations regarding their use best practices to mitigate this situation. The recommendations listed for the Kinect sensor, for example, range from advice on setting the light level of the room where the sensor is installed, as well as on the size of the area where the sensor points at, to the types of clothes that should not be worn by a player. Contrarily, public settings do not present these difficulties as they can typically be ac-

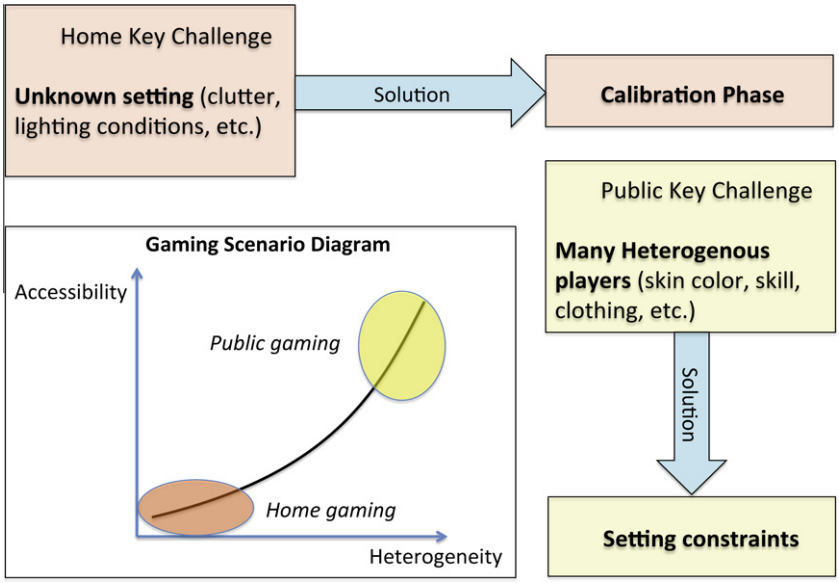

Fig. 1. A comparison between home and public gaming.

cessed, studied and tuned to meet the requirements of a particular digital game. Context accessibility, hence, is an important feature that should be exploited to optimize players' experiences within the context of public scenarios, especially when using novel gestural interaction schemes.

The heterogeneity of the players that can daily access a digital game, directly derives from the accessibility of that location, and is the second key factor that heavily differentiates public from private contexts. At home, in fact, players are in the majority of cases the same, roughly belonging to a small set of people (e.g., the components of that given household) that can access the gaming console at any given time. Therefore, the initial calibration process required to recognize some of the basic features of a new player (e.g. skin color, height, face pattern, etc.) when adopting gestural interfaces, is repeated at most a few times. In a public space, instead, a gesture-based interface needs to cope with a big variety of player types, as a random new player may begin a game once another just ended, thus making the use of an initial calibration phase, per each new person that joins the game, unfeasible. Robustness and timely responses to heterogeneity are, hence, key design requirements that emerge from this situation.

The differences between the two cited scenarios are summarized in the bottom-left graph of Fig. 1, where we can clearly distinguish those with low accessibility and player heterogeneity, which characterize the environments where home gaming consoles are installed, from those with high ones (video games displayed in public spaces), where, to the best of our knowledge, it is not possible to find well developed gesture-based interface infrastructures as well as broadly accepted design choices. The challenge, then, is to identify those algorithms that best suit scenarios where many different players, with radically different characteristics, may join a game. Such situation does not only pose a set of sub-challenges on the game itself, but also, as we shall understand from the following, on the gestural interface that mediates and interprets the input actions of a player.

3.2. From context to algorithms

Digital games played in public scenarios need, as it has become clear from the arguments that we brought in the previous subsection, to cope with high player heterogeneity, especially when new human-interaction schemes that do not require the intervention of hardware devices schemes are employed. However, public scenarios are also characterized by one advantage, that is the opportunity of defining the location and the characteristics of the location where a game will be played, i.e. the context that surrounds a game. What we will see here is that the context of a game can be exploited, along with the software and the hardware components of that game, to lead to an optimization of the interaction experience enjoyed by players.

The context that surrounds a game, hence, should not be thought as a separate entity, but as a variable that can be tuned to reach the desired performances. This means that it is possible to accurately design the location and all the characteristics of the location where a game is played, in order to facilitate the interactions that occur between a player and the virtual world where he/she moves. The size and position of the gaming arena (i.e., where the game takes place), the physical characteristics of the gaming arena (e.g., lighting, humidity, etc.), as well as the position and characteristics of the multimedia inputs/outputs are all gauges that can be adapted for this scope, depending on the particular actions and movements that a gestural interface is required to recognize. The result of this process is the definition of a *sensible area* where a player can move, while producing relevant inputs to a gaming system.

Clearly, a sensible area should meet the requirements of the particular game that is being implemented as long as those of the particular sensors that are being used. For example, when implementing a soccer game, movements should be more accurately detected close to the ground, rather than at an average person's eye height. Similarly, when using a normal webcam as a sensor, the sensible area is given by the space where the webcam is pointing at and by the illumination level of that given area. When instead using an infrared sensor, as the Wii does, the sensible area includes all those positions where the sensor can record an input from its reference light emitter, without in this case posing stringent illumination level constraints. The first important step, hence, for a game designer is that of jointly designing the context and the sensors that are employed into a particular game specification and, taking into account any further constraints that are involved, define a sensible area where a player can play. The second, and final step, is that of deriving the interaction algorithms

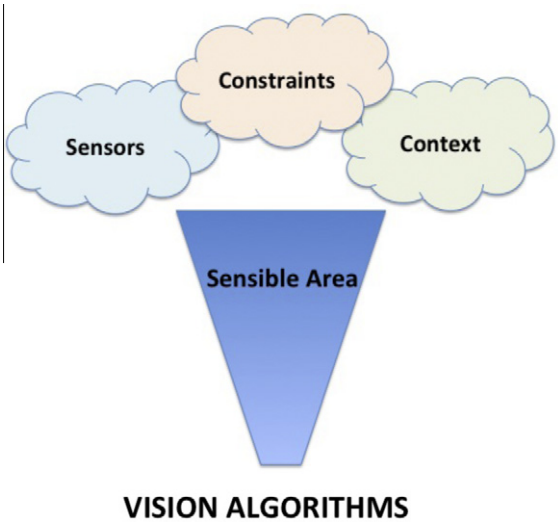

Fig. 2. Narrowing down vision algorithms.

(e.g, vision algorithms, in case the chosen sensors are video cameras) that should be employed to support efficient movement recognition within that given sensible area (Fig. 2).

Now, we wish to employ a gestural human–computer interface that responds to the constraint of using only off-the-shelf, low cost hardware. This choice responds to flexibility, low budget and deployment speed needs that other solutions, based on custom hardware, would have jeopardized. In particular, envisioning a scenario where the same gaming system may be deployed in many different locations around the world, such particular choice guarantees a fast deployment of our solution, as there is no need to ship any hardware device to support each installation, given that the intelligence of the system is solely contained within the software engine that can be simply distributed as an *entertainment app*. Last, but not least, such approach let us avoid the dependence from any proprietary hardware platform (e.g., the lock in of Microsoft Kinect).

In the top left part of Fig. 3 we show the representation of the context where a player can move his or her hands above a table. In this case, the sensible area is given by: (a) the table and its physical characteristics, (b) by our self-imposed requirement of employing off-the shelf hardware, as a simple webcam in this case, and (c) the position where the webcam is placed, i.e. above the table. In such setting, a visitor can play waving his or her hands within the sensible area (bottom of Fig. 3), as long as efficient and robust vision algorithms exist that are capable of doing so in real-time.

Let us observe that the choice of such sensible area intuitively supports an accurate tracking, and the mapping into a virtual world, of any frontal movement performed by a player's arms and hands. Hence, a simple and efficient way of implementing in software the procedure of detecting any frontal movements that are performed within this sensible area is it to extract a reference point, for example the farthest point that is reached by each hand, from the blobs that enclose a player's upper limbs. Please note that such intuitive algorithm directly descends from the definition of our sensible area and carries the advantage of not requiring any annoying initial calibration phase for each new player, as a correct light exposure and background color choice lets any simple subtraction algorithm distinguish the blob that includes the arms and hands of a player from the background. The details of how the gesture-based interface identifies, instead, the precise position of a player's hands are instead postponed to the next section.

Concluding, we have seen that the design of a gestural game for a public space requires the definition of a *sensible area*, a zone where a player interacts with a game's virtual world. More then just a zone where a player simply moves, the choice of a *sensible area* is key for the correct detection of players' actions. From the choice of the *sensible area* descends the choice of the gestural algorithms that best can work for a given set of actions. In the next section we will present a set of algorithms that fall along the sketched lines. In particular, we will show that these algorithms are particularly well suited for the *sensible area* shown in Fig. 3, as they meet the two following requirements: (a) they efficiently deal with player heterogeneity and, (b) they can be easily implemented in software (i.e., *entertainment app*), without relying on any custom hardware.

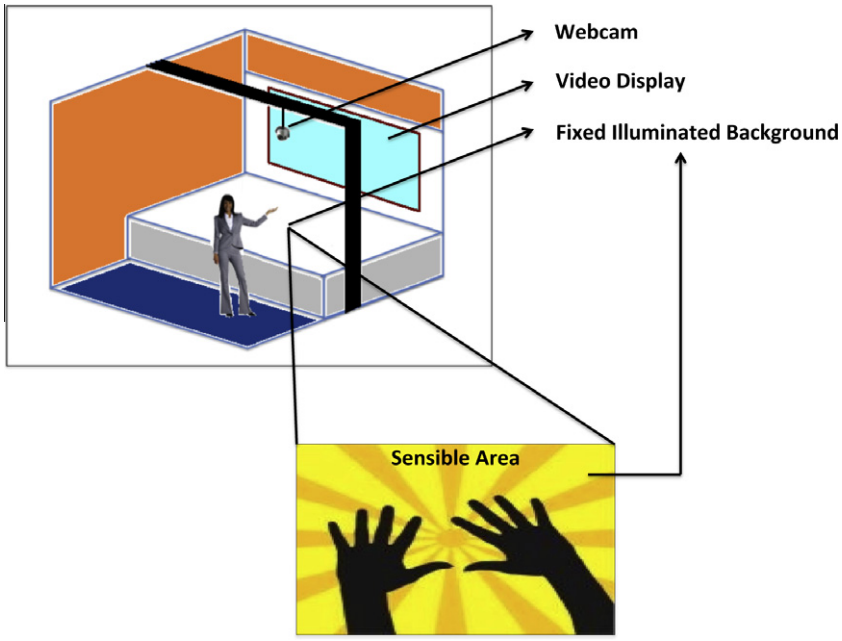

Fig. 3. Gesture-based interface scenario.

4. Algorithms

The *sensible area* that is shown in Fig. 3 has been clearly devised for the recognition of actions that occur parallel to the ground plane. Our choice, then, has been that of choosing the algorithms that could let us perform such task, with the minimum overhead and complexity. To do so, we arrived to define a cascade of three intuitive algorithms that can recognize whether an action is performed correctly or not. As a preliminary comment (confirmed also by our experimental results), these algorithms are robust and easy to implement, making them particularly suited for all those situations where gaming settings can often change or adapt to new requirements (as for example in public settings).

Anticipating now how they work, the first of these algorithms segments the areas where the upper limbs of a player lie while these move through the sensible area. This is simply done computing the luminance differences between the static background scene and that where the player has begun to move his/her hands. Once segmented, the second algorithm adopts an innovative approach, compared to those that find the center of mass of a hand, detecting all those areas that identify the farthest positions reached by a player's hands. This has been done considering that the games of our interest require a player to stretch out his/her hands in front of him/her, and hence a hand can be identified with the point that reaches farthest away from the player's body. Finally, the third algorithm recognizes the actions performed by a player following those points, based on the consideration that each movement a player performs involves a starting area from which a hand commences its movement, a trajectory to be followed, and finally an ending area where the hand completes its gesture.

4.1. Hand segmentation

The first of these algorithms performs the segmentation of the upper limbs of a player, computing the luminance differences between the static scene (i.e., the fixed illuminated background in Fig. 3) captured before anyone plays and the scene where a player has begun to move his/her hands. The static scene is saved, at the very beginning, during a very fast calibration phase, which involves dividing the sensible area rectangle into an $N \times M$ grid of blocks, and storing the maximum and minimum RGB luminance values for a subset of K pre-defined pixels within each block. N and M are chosen large enough to provide the granularity sufficient to detect a typical upper limb part of a player, whose width is about 5 cm. K, instead, is chosen as a trade-off between processing overhead (i.e. higher, for higher K values) and accuracy (i.e., lower, for lower K values). Our experiments confirmed that the values of $N = M = 9$ and $K = 4$ give optimal results in terms of both accuracy and speed. Then, for every chosen pixel p in a given block, the minimum and maximum luminance values (min_p, max_p) are taken, after an observation that lasts for a sufficient, but short, amount of time. This eliminates small luminance variations issues that are possible, even in settings subject to constant illumination. After that, with each new frame, the algorithm checks for the presence of a hand above any of the K pixels of each block. First, a low-pass Gaussian filter is applied to smoothen out any color peak due to small object movements and/or random brightness variations, providing as a result a filtered video frame. Second, for each block, a check is done to verify if the luminance value of each of the K pixels p lies within the $[min_p, max_p]$ interval of reference. If that luminance value is greater than max_p, or smaller than min_p, the pixel is considered as changed. With this comparison, the number of pixels that changed is counted for each block. If this number exceeds a given threshold (e.g., 0.75 times K), then the entire block is considered as changed. However, to be sure that the area comprised of changed blocks contains a recognizable upper limb part, all those changed blocks must be adjacent (two blocks are pair-wise adjacent if they share an edge, or even a vertex). To this aim, the final part of this algorithm checks whether all the changed blocks represent an area, a blob, comprised of all adjacent blocks, without any form of interruption between blocks. Only in such case the segmentation process ends successfully, returning a blob that includes a player's upper limb. Fig. 4 delivers a pictorial representation of this process.

4.2. Hand recognition

Once the blobs containing the upper limbs have been segmented, the problem is to find an efficient mechanism to identify and follow a player's hands. Achieving such result means determining one relevant point to follow for each of the two blobs. Some authors, in similar cases, choose the center of mass; we, instead, as relevant point to follow for each hand, have chosen the *extreme point*, the point that reaches the farthest position of a player's hands, while waving them freely in the air. Many games, in fact, require players to extends their hands in front of them, and hence a relevant information for following, and then for recognizing, gestures is concerned with that precise point that reaches farthest away from the player's body. Finding both of the extreme points (right and left hands) can be performed utilizing the following algorithm. We start by mapping the sensible area, within which hands are waved, into a bi-dimensional Cartesian coordinate system. In this coordinate system a unit of length is taken as the linear dimension of a block. The x and y axes have their origin at the bottom leftmost corner of the surface, where y points externally away from the player's body, while x points towards the right side of the surface, as shown in the top-left graph of Fig. 5. In the first step of this algorithm, each blob is mapped into a bar chart, whose height y equals, for a given x value, to the farthest distance of that blob from the player's body. Simply said, as shown in the leftmost chart at the bottom of Fig. 5, each bar represents the highest y value (for each x body's position), where an upper limb part, was detected. With the second step of the algorithm (that is, determining the coordinates of the extreme point of the first hand) that point is found by that bar with the maximum y value among all the represented bars in the chart. For instance, in our example, the extreme point of the first hand is detected at position $x = 12$, with value $y = 7$. Trickier is the problem of finding the extreme point of the second hand, as it cannot be individuated in correspondence with the position where the second (or third) highest y value is found in the chart. These, in fact, could correspond to local maximum values that represent lower points of the first hand. In essence, we are seeking for another maximum y value, while excluding all those bars that still pertain to the first recognized hand. Thus, with the aim of filtering out all these possible *fake* maximum values, our algorithm constructs a new bar chart, based on the first one, where each new bar is created as follows. The height (y) of each bar of the old chart is contrasted with a so-called *minimum* (whose value is progressively updated). We start with the next bar at the left (or at the right) of that for which the global maximum was found. If the difference between the height of this current bar and the *minimum* is positive, then a new bar is inserted in the new chart, whose height equals the value of their difference. If the difference is negative, instead, the bar is not imported in the new chart. Obviously, the *minimum* represents the y value of the lowest bar encountered so far, and is updated as long as the search proceeds, with each new bar under consideration. This methodology applied to the leftmost chart of Fig. 5 yields the correspondent rightmost chart of the same Figure. In fact, if we move along the direction of decreasing x values, we obtain a new chart where at position $x = 11$, we have $y(11) = max(5 - 7, 0) = 0$, as the minimum y value

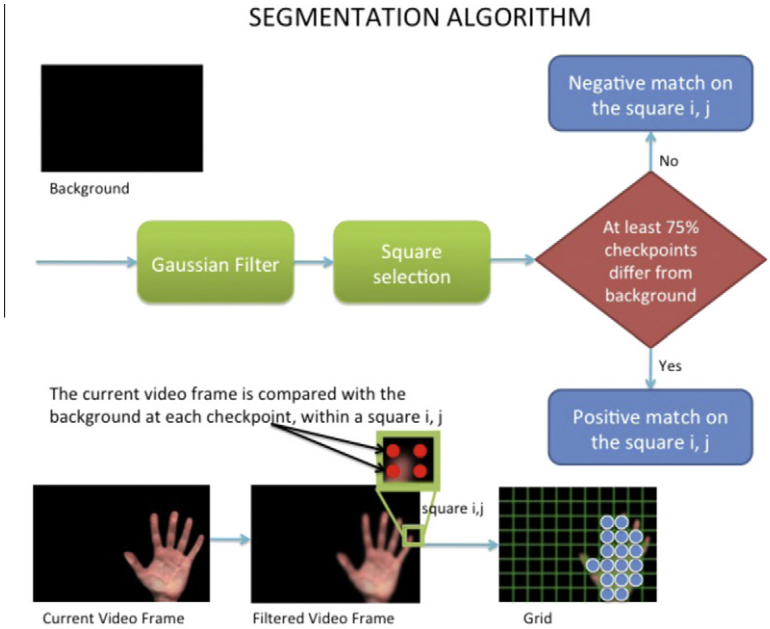

Fig. 4. Upper limb segmentation steps.

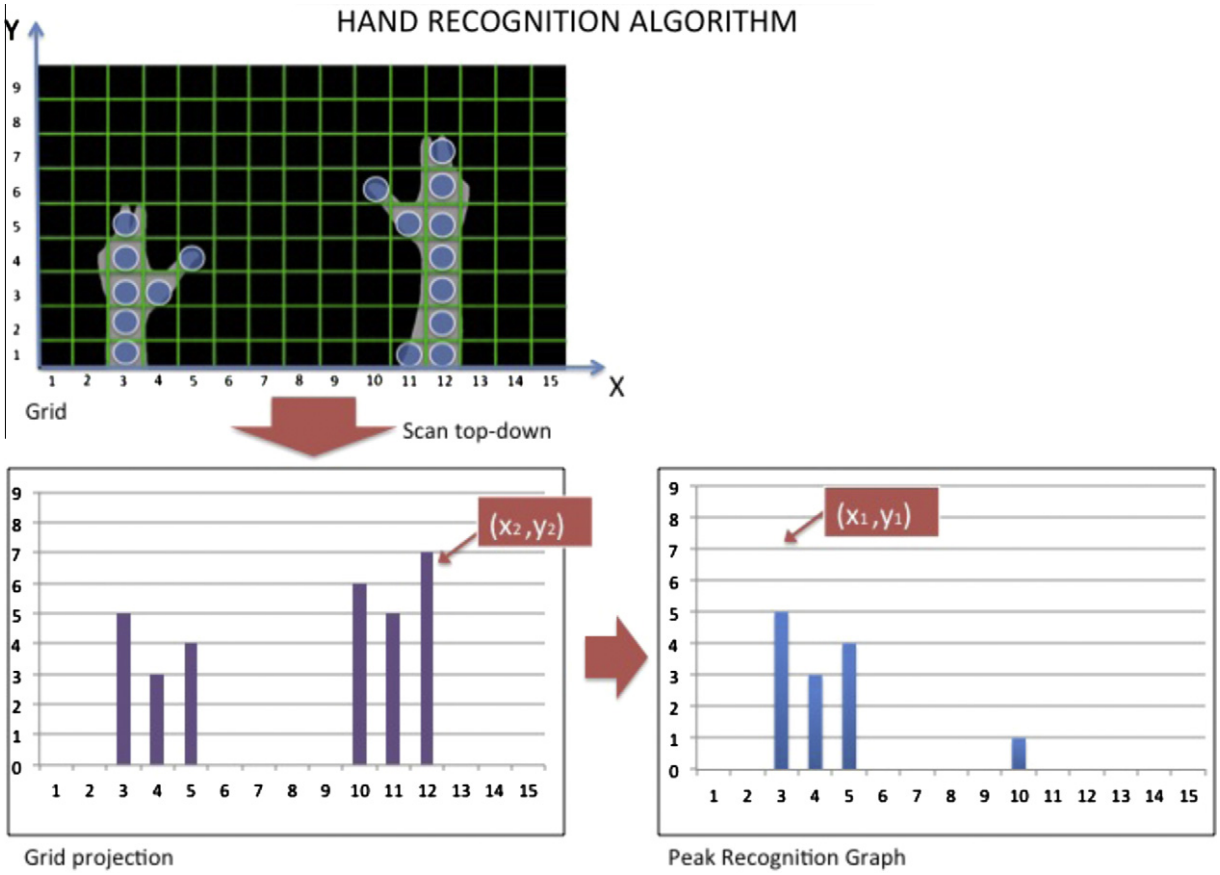

Fig. 5. Hand recognition steps.

encountered so far corresponds to the value of the global maximum, that is 7. After that, the new minimum y value is updated to 5, and hence for the position $x = 10$, we obtain a new bar in the new chart, whose y value is equal to $max(6 - 5, 0) = 1$. Again, at the position $x = 5$, the minimum value computed until that point is 0 and hence $y(5) = max(4 - 0, 0) = 4$. If we iterate this until the entire old chart is completely scanned, a new chart is produced with $y(4) = 3$, $y(3) = 5$, respectively. On the new chart, a new global maximum can be searched. That new maximum y value corresponds to the extreme point of the second hand (considering that

only one player plays at a time). In our example, it is identified at the position $x = 3$. With the extreme point of each hand, we can now follow hands and their movements.

4.3. Action recognition

To recognize actions, we exploit the fact that each gesture requires a player to move his/her hand between an origin and a destination, along a given trajectory connecting any two given areas within the sensible area. Moreover, in many cases, an action that involves following a given trajectory must be done within a given time period, after which that movement has been performed too slowly to be considered correct. This is true for all the movements that a player is required to perform within a very large gamut of games where hands can be moved freely. Hence, the idea behind our algorithm is that of tracking the extreme point of each hand, while verifying that this point starts its movement from the origin of a given action, completes it in correspondence with the destination, and traverses a set of checkpoints set along the chosen trajectory (see Fig. 6). Therefore, each trajectory followed by an extreme point is recognized as correct if it flows within a stripe of a given size (in essence, a certain degree of tolerance is allowed). This mechanism was devised to make our algorithm able to consider as correct a wider set of movements with slightly different trajectories that differentiate only for a few geometric differences. This scheme can be applied to trajectories of either linear or circular shape.

4.4. A comparison with other approaches

It is now important to stop and understand whether the algorithms that we have presented in this section really represent the best choice for a situation like the one depicted in Fig. 3, or, alternatively, other better available solutions exist. In order to do so, we will now report on the results drawn from an experimentation campaign that aimed at comparing two additional solutions, chosen among those discussed in Section 2, to our approach.

We carried out this comparison measuring how often an action performed by a pool of players was correctly detected by the three different schemes. We did not need to perform other types of tests, in this phase, as it is straightforward that our approach is the best in terms of: (a) cost, not requiring any custom or proprietary hardware, and (b) efficiency, since it avoids any initial calibration phase and it is based on algorithms with little computation power demands.

In particular, we identified three types of movements that could be performed by a player's hands within the sensible area shown in Fig. 3. The first is a simple movement where a hand moves from one side to the other of a 1-m wide table. The second, instead, includes two movements of the first type, with the additional complexity given by a change of direction. Finally, the third, which is also the more complex, is a circular movement, hence a movement that begins and ends upon the same spot. Three different people repeated all three movements ten times at a slow, normal and high speed, respectively.

The same tests were carried out utilizing three different gestural interfaces: the first was based on the cascade of algorithms described in Sections 4.1–4.3, which are here termed SensArea, the second on the Nintendo Wii and the third on the OpenCV libraries. The results shown in Fig. 7 meet what we expected: the Nintendo Wii based solution prevails overall in terms of robustness, closely followed by SensArea that achieves satisfying results. The OpenCV-based one, instead, placed itself at the third place. It is worth spending a few more comments, on the comparison between our SensArea solution and the OpenCV one. While the performance results they exhibit are very close, from our approach two particular advantages emerge over the algorithms that are utilized within OpenCV: (a) it does not require an initial calibration phase due to the estimation of the skin color of a player, per each player, and (b) the algorithms it employs are of a very low complexity, which run in constant time.

5. A real assessment: the case of Tortellino X-Perience at the 2010 Shanghai World Expo

We practically performed the assessment of our algorithms, implementing a new game, termed *Tortellino X-Perience*, that was specifically thought for public spaces (the public space context

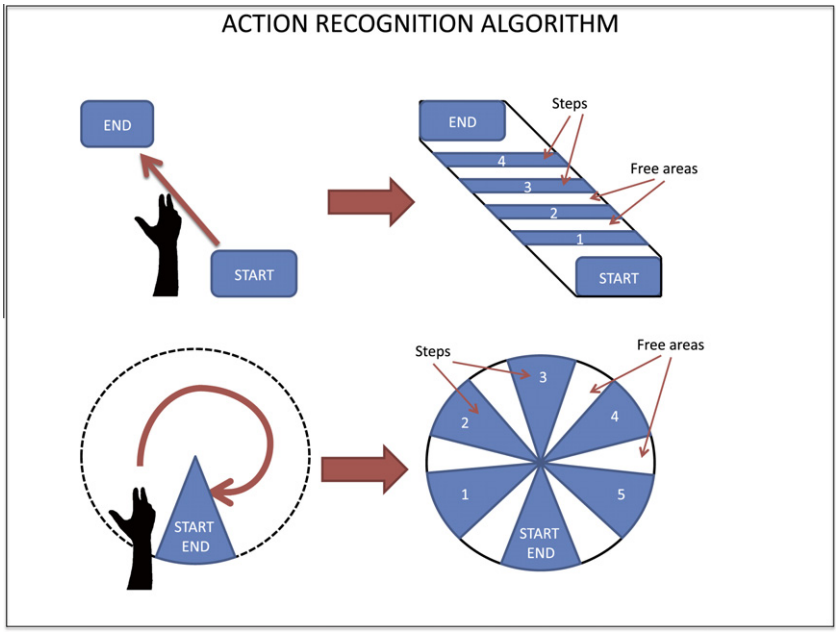

Fig. 6. Recognizing actions.

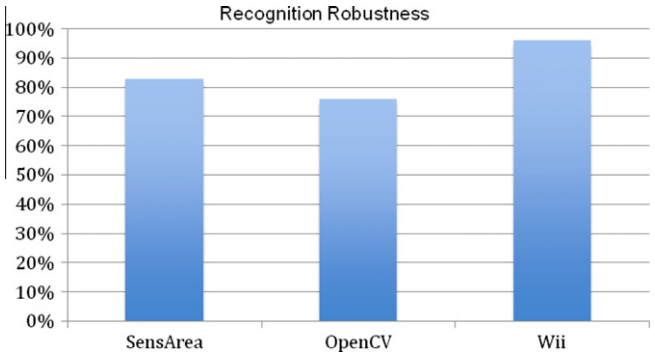

Fig. 7. Action recognition algorithms robustness.

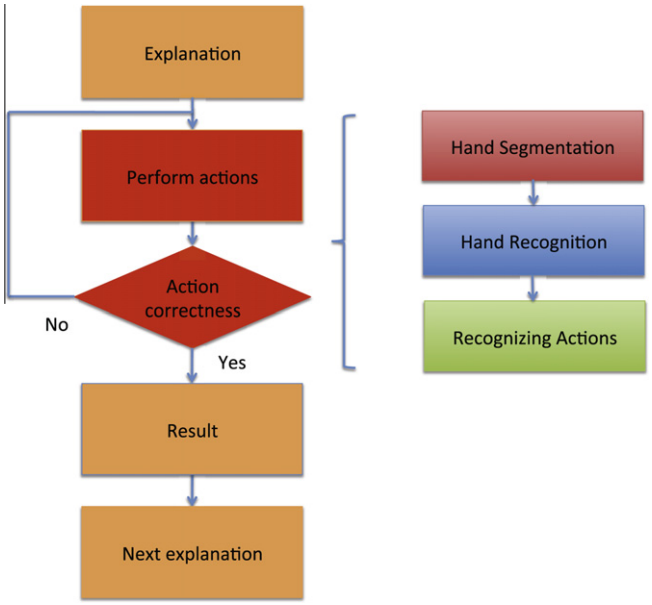

Fig. 8. Flow diagram depicting how a single phase of the game works.

and the sensible area matched the one depicted in Fig. 3). Before getting to how it works and to its performance, let us anticipate that this game has been tested by several hundreds of people visiting a game stand in Shanghai, at the 2010 World Expo, at the Cantina Bentivoglio (a renown and appreciated restaurant of Bologna more recently in June 2011) and at the Bologna Town Hall (in occasion of the international Tortellino chef contest) [3,4,22].

Tortellino X-Perience represents an example of a game that, displayed in public spaces, requires the use of a gestural interface. The scope of the game is that of challenging a player at performing the same actions that a real cook would do when preparing a Tortellino (i.e., a traditional pasta dumpling of the city of Bologna).

5.1. How it works

The game evolves in phases, where within each phase a player, at first watches a video explanation of how a given part of the recipe should be carried out, and then performs the actions that compose that part of the recipe within a sensible area that is located on top of a table that lies in front of him/her, just as in a real kitchen. A video camera captures those actions and provides the algorithms discussed in Section 4 with the information necessary to decide on their correctness. While the player moves, a representation of his/her hands is shown on the screen, within a virtual kitchen. The sequence of these steps is summarized in the flowchart of Fig. 8.

To better exemplify the main steps of the flowchart of Fig. 8, namely, the *Explanation* step, the *Perform action* step and the *Action correctness check* step, we report in Fig. 9 three subsequent frames taken from a video shot during the World Expo in Shanghai that witness how the evolves through the three aforementioned steps. Specifically, the leftmost part of Fig. 9 shows a frame of the video explanation, which tells the player how a ball of dough should be kneaded. The central part of Fig. 9, instead, depicts a player while he kneads the ball of dough, moving his hands in the sensible area. Finally, the rightmost part of Fig. 9, displays how the virtual cooking board appears to the player while he performs his actions.

5.2. Game rules

The different phases of the *Tortellino X-Perience* game are discussed below in isolation, along with a snapshot taken from the reality and its corresponding storyboard representation.

Phase 1. *Move the ingredients at the center of the cooking board*: The cooking board is empty, while flour and eggs appear at its right hand side. The player is required to move the ingredients at the center of the cooking board (Fig. 10, left: photo, right: storyboard);

Phase 2. *Knead the ingredients*: the yolks and the albumen of the opened eggs float squeezed within the flour (Fig. 11, left: photo, right: storyboard). The required action is to knead the ingredients, adding some water, ending up with a smooth ball of dough;

Phase 3. *Roll the dough*: the ball of dough is sitting over the cooking board. The dough, at this point, needs to be rolled with a rolling pin (Fig. 12, left: photo, right: storyboard);

Phase 4. *Cut the dough*: a thin foil of dough is lying on the cooking board. This should now be cut in squares, using a cutting wheel (Fig. 13, left: photo, right: storyboard);

Phase 5. *Stuff the Tortellini*: Now, each square of dough should be filled with the ingredients (meat and cheese) that sit at the side of the cooking board (Fig. 14, left: photo, right: storyboard);

Phase 6. *Close the Tortellini 1*: each pasta dumpling, now open, needs to be closed joining the first pair of opposite angles (Fig. 15, left: photo, right: storyboard);

Phase 7. *Close the Tortellini 2*: Each half-closed Tortellino becomes ready to be cooked after closing the second pair of its opposite angles (Fig. 16, left: photo, right: storyboard).

5.3. Performance and survey results from a public exhibition

The demonstration of this game in public gave us the opportunity of testing its efficiency over a very large and diversified sample of people, providing us with a good set of results. A first set of experiments assessed the speed of the cascade of our algorithms, measuring the time taken by the three algorithms to return a correctness decision on a given action performed by a player. These results are shown in Fig. 17, where we can appreciate that on over 130 events detected by our algorithm, less than 4% required more than a 20 ms processing time, never, however, exceeding a 30 ms delay.

A second set of results, instead, has been acquired asking players to answer a few survey questions, in particular:

1. Did you enjoy the game?
2. Was it easy and intuitive to play?
3. Would you have preferred playing with a remote control?

Fig. 9. Left: kneading the dough, center: playing the game and right: virtual cooking board.

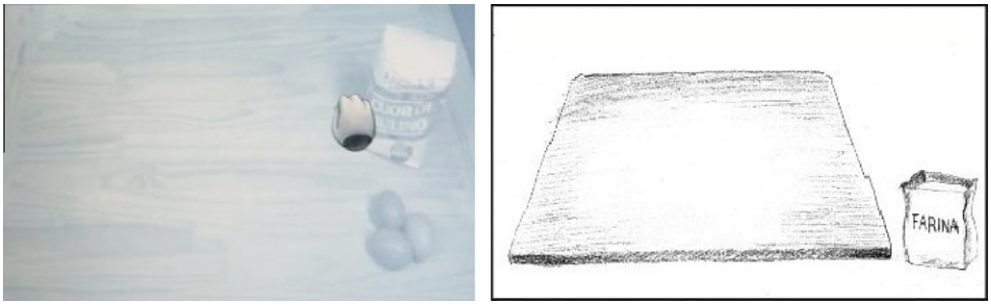

Fig. 10. The cooking board is empty.

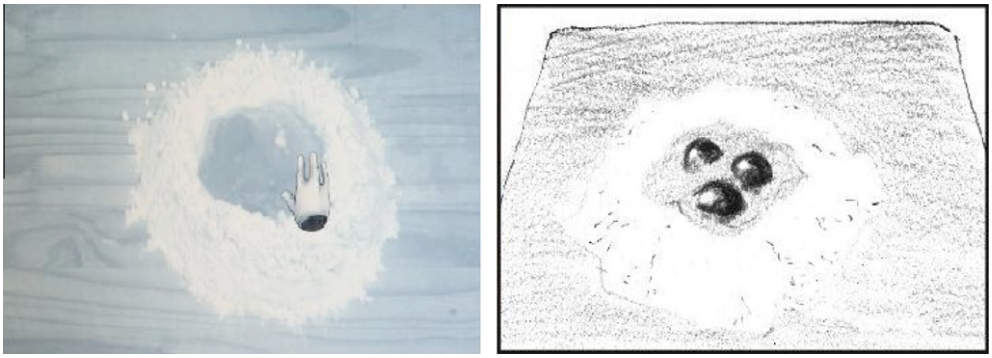

Fig. 11. The eggs float squeezed within the flour.

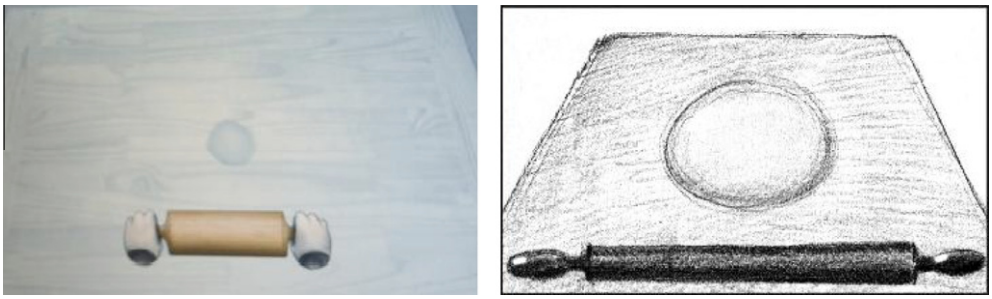

Fig. 12. The ball of dough needs to be rolled.

Players were asked to give a score, between 1 and 5, to answer the first two questions, while they were asked to simply give a positive or negative answer to the last one. The outcome of this survey is reported in Figs. 18–20 (answers are grouped by the age ranges of participants). Fig. 18, which reports the score, by age range, assigned as an answer to the first question, clearly shows there has been a high appreciation for the game, which has especially been enjoyed by older people, we believe because of its theme. Fig. 19, instead, shows that, when answering the second question, there has been no significant difference between people of different ages: all players have appreciated the gestural interface, on average. This fact has also been confirmed by the answers we received to the third question, summarized in Fig. 20, where most players clearly affirmed that they preferred our hands-free

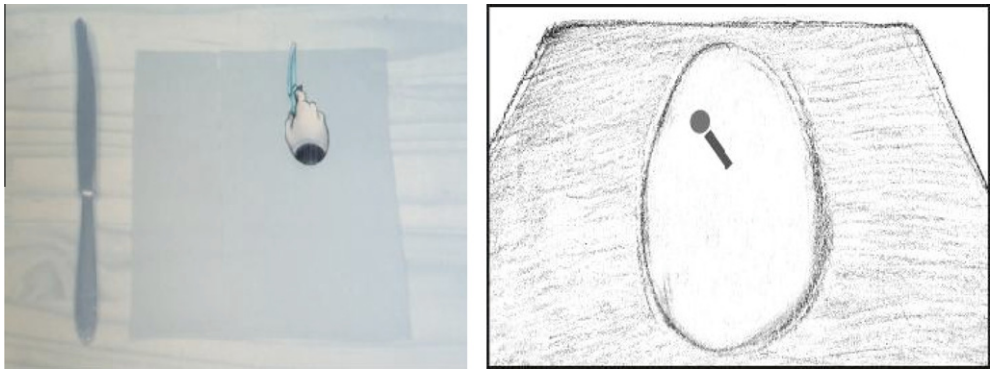

Fig. 13. The foil of dough waits to be cut.

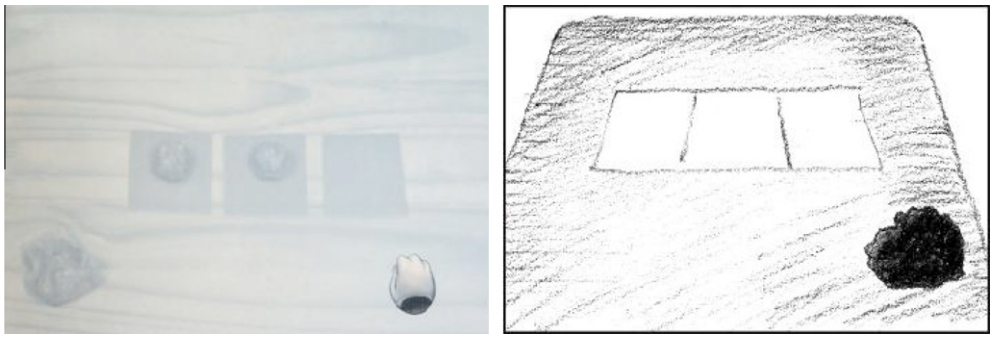

Fig. 14. Filling the squares of dough.

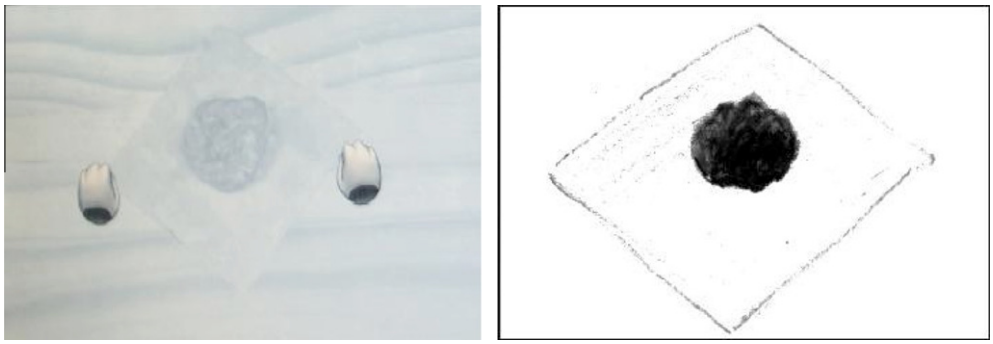

Fig. 15. Closing the squares of filled dough.

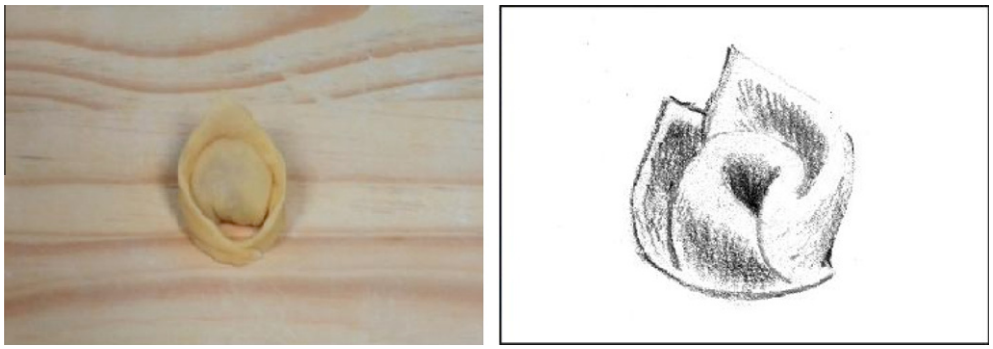

Fig. 16. Ready to be boiled.

interaction scheme to the use of a remote control. Interestingly, we obtained a few preferences for the use of a remote control among players whose age ranged between the late thirties to the late forties; therefore all people belonging to the generation that most has

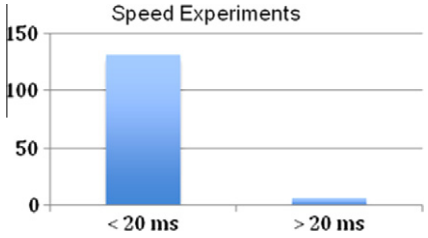

Fig. 17. Speed results.

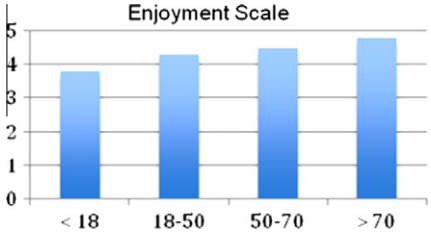

Fig. 18. Survey results: enjoyment by age range.

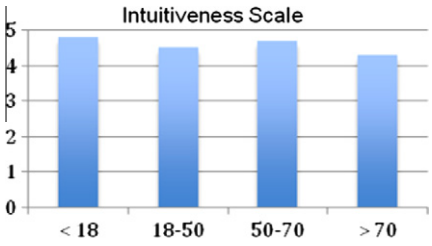

Fig. 19. Survey results: intuitiveness by age range.

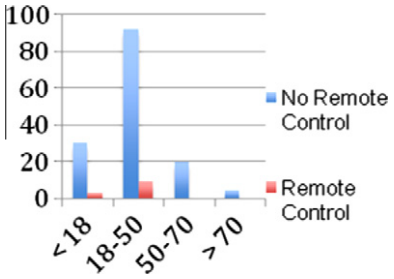

Fig. 20. Survey results: preferring a remote control by age range.

used, and hence feels most comfortable with, such type of hardware interface.

6. Broadening the field of application

One interesting issue is that of understanding if our gesture-based interface may be adapted to a broader gamut of games, where the movements are different from those treated in the game (Tortellino X-Perience) we developed and discussed in the next section. Even more interesting is to realize if our gestural interface can also provide support to a larger set of performing events (e.g., perhaps artistic or scientific), spanning out of the gaming field. One could, rightly argue that our system, as such, does not comply with all the requirements that a generic performing event can display. The reason for this is that the algorithms we have so far defined, as they are, do not explicitly address some known challenges in object recognition. Problems may arise, for example, when dealing with underlying background clutter or with situations where the upper limbs of a player become partially or temporarily occluded. Moreover, these algorithms have been designed thinking at a single player scenario, being statically locked on the recognition of the actions that one or two hands perform while moving within a given sensible area.

In reality, as we shall shortly discuss, it is possible to broaden the spectrum of application of our gesture-based interface by simply adapting the algorithms to successfully deal with all of the mentioned problems. We will touch upon each single challenge, below in isolation.

6.1. Dealing with occlusions and multiple players

While our schemes have not been explicitly devised to deal with occlusions, they can be easily extended in such sense. With a single player an occlusion can occur, for example, if a player joins or overlaps his or her hands, thus leading the system to detect one hand instead of two. Occlusion might also happen if another person entered the sensible area and disturbed a game while this was played, for example taking one of the player's hands. But an intruder could also act maliciously, for example extending his/her hands to take control of the game.

While occlusions and multiple players represent two different problems, we can envision a single solution for both. Before describing the solution, we should remind that in a controlled setting as a public exhibition, the sensible area of a game can be decided in a way that occlusions never occur. Nonetheless, for the sake of completeness, let us now assume that at some point during a game, one of the two following events happen: (a) a player joins his upper limbs starting from his or her elbows, letting them overlap, or (b) another person enters the sensible area, taking control of the ongoing game. In the case (a) happens, the trajectories of the two extreme points (i.e., where one extreme point identifies one hand) will converge until the two points merge into a single one. Such problem can be solved observing that physical objects cannot suddenly appear or disappear. Hence, to deal with (a), and in general with any type of occlusion, the algorithm should be extended to save the evolution in time of the trajectory of extreme points. Hence, if (a) happened, the algorithm could detect the convergence of the two trajectories, and also detects that a single extreme point individuates both hands.

Now, a variation of the just discussed mechanism can be used to deal with (b), case where a malicious player jeopardizes a game extending his/her upper limbs. This extension is composed of two parts; the first reveals the hands of multiple persons, while the second associates the hands with a player profile. The first part of the extension simply entails searching for all the local maxima that exceed the corresponding local minima, by a given threshold. Hence, instead of stopping our algorithm after the second hand is individuated, this can be augmented to search for further pairs of hands that may be waving within the sensible area. The second part of the extension, instead, profiles the extreme points (i.e., hands) that are revealed, associating them with a player. In this way it is possible to know whose hands are moving within the game, hence disable those actions that are performed by intruders, for example. Tracking and profiling the extreme points of a player, hence, it is possible to keep track of the player that is rightfully playing, sorting out any intruder.

All this holds for the specific case we have treated so far; obviously, situations may occur where a more general kind of occlusion can interfere with the recognition of actions performed by hands. Think for example of a generic object interfering with the hand recognition process. Even though the occurrence of these facts should be excluded by the particular setting imposed to the sensible area, nevertheless our gesture recognition interface may be extended

Fig. 21. Catching candies.

Fig. 22. Playing drums.

and re-enforced to overcome this problem with traditional occlusion removal techniques, as those described in Section 2 [19,20].

6.2. Dealing with arbitrary actions

The gesture-based interface that we have so far defined well suits all those games where the actions performed in the space delimited by a sensible area occur. Given the flexibility of such system, once the actions that should be performed within a given game have been sketched, a gesture-based interface can be implemented simply: (a) placing a webcam and hence defining the sensible area where actions occur, and (b) defining those actions within our action recognition algorithm, simply providing their order, timing, trajectory, initial and final positions. Now, all this does not require any modification to how the interface is implemented, as it only depends on external configuration files that can be simply overwritten, depending on the game that is being supported.

An example of a game that could be easily supported by our gesture-based interface is provided in Fig. 21, where a player is required to catch the candies that are falling from the sky. In this particular case, there is no need to recognize a given action, as the game only depends on the position, at a given time, of the hand that holds the box. The software layer that implements the game, hence, does not depend on the recognition of a particular action, but on the coincident position, at a given time, of the box and a given falling candy. The game shown in Fig. 22, differs from the preceding one, since in this case it is possible to identify fixed final positions of interest to the game. These positions are given by the location of the drums. When touching any of the drums an action is in this case recognized, and this information is provided to the game logic. Finally, another more complex case is given by the game sketched in Fig. 23, where a player is asked to push items into their boxes as they fall from the sky. In this case, as in that of Fig. 21, the game logic requires to know the position of a player's hand, to detect when a falling item is hit. However, not only the position, but also the direction from where the player hits the item, here, has its importance. Our recognition algorithm can then be set to detect when the player is hitting a item from the right direction, from left to right in Fig. 23.

6.3. Dealing with clutter

Background clutter can lead to difficulties in recognizing patterns, as a messy background can confuse a segmentation algorithm, leading to its failure in detecting a given pre-defined shape. Such problem accrues with objects that can change their shape in time, as for example a hand can (e.g., open vs. closed hand). In our case, however, the background provided by a sensible area is one of those variables that a game designer can adjust to improve the performance of the entire gesture-based interface. Clutter, hence, can be completely removed or be set to vary very slowly, according to a game's requirements. Under such conditions,

Fig. 23. Pushing items into their boxes.

our algorithm can learn the underlying background, at a given speed, storing the ranges of the luminance values observed at the $K \times N \times M$ key points that are periodically analyzed to segment the upper limb of a player. Hence, a game designer is given the chance to let the background change (e.g., for aesthetic reasons an underline video could be projected while the game is active), as the segmentation of a player only depends on limited groups of pixels that include the key points that are analyzed by our algorithms.

7. Conclusion

We showed that a set of algorithms exists that, opportunely customized, meet the requirements that gaming in public spaces impose. To demonstrate this, we realized a set of algorithms that are specifically designed for gesture-based interfaces for public spaces. We have shown, in this work, that it is possible to reach very good performances, in terms of gaming experience, solely relying on a simple video camera and on a robust gesture recognition software system. The efficacy of our approach has been demonstrated through the design and implementation of a novel game, Tortellino X-Perience, which has been enjoyed by hundreds of visitors at the 2010 Shanghai World Expo and at the Cantina Bentivoglio in Bologna in 2011.

Acknowledgment

The authors acknowledge the International FIRB Damasco Project for supporting this work.

Appendix A. Implementation details

At the very initial stage, before anyone plays, a unique calibration phase is performed to store the fixed background image. Such phase entails taking a picture of the sensible area at selected key points (pixels) and storing the maximum and minimum color values per each point p, $[min_p, max_p]$. After this initial phase a Gaussian filter, required to smoothen any color peak values given by

noise, is applied to video frames in real-time. The image representing the fixed background is then logically divided into squares, where each square contains *N* key points. For each square, the algorithm checks each single key point, and counts how many key points take a color value that lies out of the stored color range that represents the fixed background. If, within a square, at least 75% percent of the pixels take a value that does not fall in the stored color interval, then a positive match is returned for that given square.

Once the segmentation phase terminates, the hand recognition algorithm steps in, beginning with a search for the farthest ahead position (i.e., closest to the screen) that is reached by a player at a given time. This operation is performed by the `scanTopDown` function (line 1 of Fig. 24) that saves the maximum row value where a relevant background difference has been detected, per each column, in the projection data structure. In case the maximum row value stored in projection exceeds a given pre-defined threshold (line 2 of Fig. 24), the algorithm saves this value as the current position of the first hand in the variable pair (`max_index`, `aux_min`) (lines 3 and 4 of Fig. 24). The problem, at this point, is to verify whether both of the player's hands are waving over the gaming area. Such operation is performed searching for the second maximum row value of the blobs in both of the semi-planes located at the right and the left of where the first hand has been found, hence iterating for `i` values that are greater and smaller than `max_index` (lines 5 through 12 of Fig. 24), searching for the highest local maximum value `peak[i]` that exceeds the so far encountered minimum value, stored in the variable `aux_min`, by a given threshold stored in `tolerance` (lines 6 and 18 of Fig. 12). If only one hand was present on the gaming arena, the algorithm returns a single coordinate pair (line 14, 19 and 20 in Fig. 24), while the opposite situation is dealt with in lines 16 to 20 of Fig. 24.

The action recognition algorithm is the final step that is necessary to decide whether a relevant action has been performed or not. This algorithm is based on the intuitive observation that each action requires a player to move his or her hand between an origin and a destination point. This is performed implementing the pseudo-code in shown in Fig. 25. Actions are basically broken down in the steps that compose a trajectory, and periodically our algorithm checks whether a given player is following the right steps. When an action begins (line 1 of Fig. 25), our algorithm at first checks that the player's hand is placed within the correct initial position, if not an error message is returned (lines 2–4 of Fig. 25). In case the player started from the right position, our algorithm checks that a player moves following the right steps (lines 5 through 11 in Fig. 25) and the right direction (lines 12 through 16 of Fig. 25). If, eventually, the player gets to the final position the action is completed (lines 17–19 of Fig. 25). Clearly, timing plays an important role in actions. For the sake of clarity, although we implemented a timeout mechanism that checks for the correct timing of actions, we avoided reporting on such aspect in our pseudo-code of Fig. 25.

```
1.  projection= scanTopDown(grid);
2.  if(max(projection) > tolerance) {
3.    max_index = indexOf(max(projection));
4.    aux_min = max(projection);
5.    for (i = max_index+1; i < x.length; i++){
6.        aux_min = min(projection[i], aux_min);
7.        peak[i] = projection[i] - aux_min;
8.    }
9.    aux_min = max(projection);
5.    for (i = max_index-1; i> = 0; i--) {
10.       aux_min = min(projection[i], aux_min);
11.       peak[i]= projection[i] - aux_min;
12.   }
13.   if(max[peak] < tolerance)
14.       x1 = x2 = max_index;
15.   else{
16.       x1 = min(indexOf(max(peak)), max_index);
17.       x2 = max(indexOf(max(peak)), max_index);
18.   }
19.   y1 = projection[x1];
20.   y2 = projection[x2];
21. }
```

Fig. 24. Hand recognition steps.

```
1. if(actionStarted){
2.     if (!actionArea.contains(handPosition)){
3.        error("HAND POSITION OUT OF BOUNDS");
4.     }
5.     else {
6.        for(i = 0; i < steps.length; i++){
7.           if(steps[i].contains(handPosition)){
8.              if (i == currentStep ||i == currentStep + 1){
9.                 currentStep = i;
10.                break;
11.             }
12.             else{
13.                error("WRONG STEP ORDER");
14.             }
15.          }
16.       }
17.       if(currentStep == steps.end()){
18.          ok("ACTION COMPLETED");
19.       }
20.    }
21. }
22. else if (steps[0].contains(handPosition)){
23.     actionStarted = true;
24. }
```

Fig. 25. Recognizing actions: pseudocode.

References

[1] M. Roccetti, G. Marfia, Recognizing intuitive pre-defined gestures for cultural specific interactions: an image-based approach, in: Proc. 3rd IEEE International Workshop on Digital Entertainment, Networked Virtual Environments, and Creative Technology, Las Vegas, 2011.

[2] M. Roccetti, G. Marfia, M. Zanichelli, The art and craft of making the Tortellino: playing with a digital gesture recognizer for preparing pasta culinary recipes, ACM Comput Entertain. 8 (4) (2010).

[3] Bologna Shanghai World Expo. <http://www.bolognaexpo2010.it/zh20/tortellino-x-perience> (accessed 20.06.11).

[4] President from University of Bologna visited Tongji, Tongji University (website). <http://www.tongji.edu.cn/english/classid-61-newsid-32897-t-show.html> (accessed 20.06.11).

[5] T.B. Moeslund, A. Hilton, V. Krüger, A survey of advances in vision-based human motion capture and analysis, Comput. Vis. Image Understand. 104 (2–3) (2006) 90–126.

[6] S. Mitra, T. Acharaya, Gesture recognition: a survey, Trans. Syst. Man Cybernet. 37 (3) (2007) 311–324.

[7] T. Selker, W. Burleson, Context-aware design and interaction in computer systems, IBM Syst. J. 39 (3.4) (2000) 880–891.

[8] R.Y. Wang, J. Popovic, Real-time hand-tracking with a color glove, ACM Trans. Graph. 28 (3) (2009) 1–8.

[9] A. Rhalibi, M. Merabti, P. Fergus, S. Yuanyuan, Perceptual user interface as games controller, in: Proc. 5th IEEE Consumer Communications and Networking Conference, Las Vegas, 2008, pp. 1059–1064.

[10] C. Harrison, A.K. Dey, Lean and zoom: proximity-aware user interface and content magnification, in: Proc. 26th Annual ACM SIGCHI Conference on Human factors in Computing Systems, Florence, 2008, pp. 507–510.

[11] D. Bannach, O. Amft, K.S. Kunze, E.A. Heinz, G. Troster, P. Lukowicz, Waving real hand gestures recorded by wearable motion sensors to a virtual car and driver in a mixed-reality parking game, in: Proc. IEEE Symposium on Computational Intelligence and Games, Hololulu, 2007, pp. 32–39.

[12] S. Ferretti, M. Roccetti, Fast delivery of game events with an optimistic synchronization mechanism in massive multiplayer online games, in: Proc. 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology, Valencia, 2005, pp. 405–412.

[13] M. Roccetti, P. Salomoni, A web-based synchronized multimedia system for distance education, in: Proc. 2001 ACM Symposium on Applied Computing (SAC '01), Las Vegas, USA, pp. 94–98.

[14] D. Campbell, Physical gaming: out of the lap and into the living room, in: Proc. 7th IEEE/ACM International Symposium on Mixed and Augmented Reality, Cambridge, 2008.

[15] M. Pasch, N. Bianchi-Berthouze, B. Van Dijk, A. Nijholt, Movement-based sports video games: investigating motivation and gaming experience, Entertain. Comput. 1 (2) (2009) 49–61.

[16] J.C. Lee, Hacking the Nintendo Wii remote, IEEE Pervasive Comput. 7 (3) (2008) 39–45.

[17] T. Schlomer, B. Poppinga, N. Henze, S. Boll, Gesture recognition with a Wii controller, in: Proc. 2nd International Conference on Tangible and Embedded Interaction, New York, 2008, pp. 11–14.
[18] A.D. Wilson, Depth-sensing video cameras for 3D tangible tabletop interaction, in: Proc. 2nd Annual IEEE International Workshop on Horizontal Interactive Human–Computer Systems, Newport, 2007, pp. 201–204.
[19] M.S. Arulampalam, S. Maskell, N. Gordon, T. Clapp, A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking, IEEE Trans. Signal Process. 50 (2) (2002) 174–188.
[20] J. Illingworth, J. Kittler, A survey of the Hough transform, Comput. Vis. Graph. Image Process. 44 (1) (1988) 7–116.
[21] C. Manresa, J. Varona, R. Mas, F.J. Perales, Hand tracking and gesture recognition for human–computer interaction, Electron. Lett. Comput. Vis. Image Anal. 5 (3) (2005) 96–104.
[22] D. Mitzman, Italian Hi-Tech Software Teaches Perfect Pasta Skills, BBC News (06/22/11). <http://www.bbc.co.uk/news/world-europe-13856559> (accessed 20.06.11).
[23] J. Robertson, The Wonderful World of Cooking Interfaces. Computers, Creativity and Learning (Blog). <http://judyrobertson.typepad.com/judy_robertson/2011/02/the-wonderful-world-of-cooking-interfaces.html> (accessed 20.06.11).
[24] J. Giles, One per Cent: Video Games Teaches You To Make the Perfect Tortellini, New Scientist, January 2011. <http://www.newscientist.com/blogs/onepercent/2011/01/computer-game-that-teaches-you.html> (accessed 20.06.11).
[25] C.E. Palazzi, S. Ferretti, S. Cacciaguerra, M. Roccetti, A RIO-like technique for interactivity loss avoidance in fast-paced multiplayer online games, ACM Comput. Entertain. 3 (2) (2005) 1–11.