Design and implementation of a flexible hand gesture command interface for games based on computer vision

João L. Bernardes[1] Ricardo Nakamura[2] Romero Tori[1]

[1,2]Escola Politécnica da USP, PCS, Brazil [1]Centro Universitário SENAC, Brazil

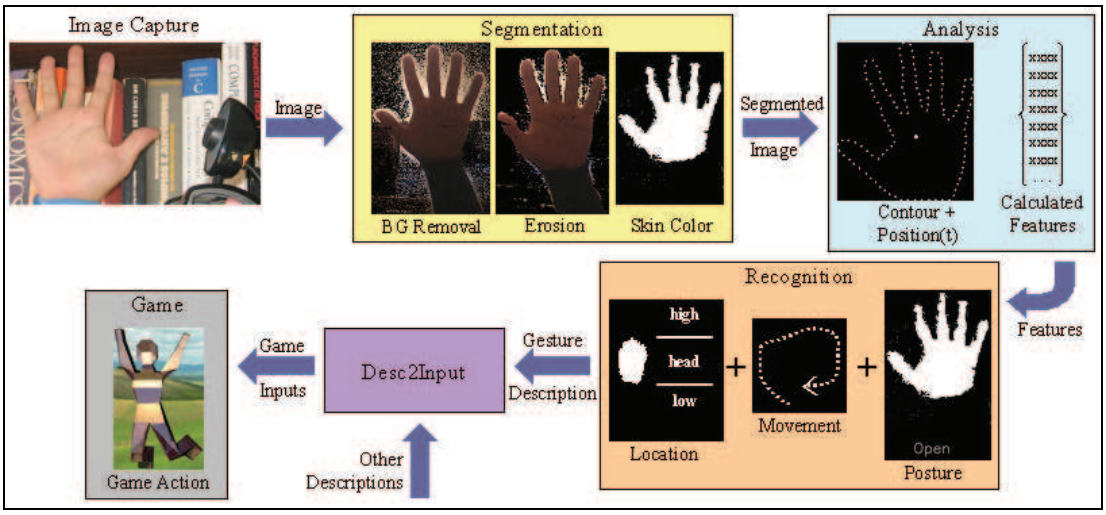

Figure 1: Gestures2Go

Abstract

This paper describes a command interface for games based on hand gestures defined by postures, movement and location. The large variety of gestures thus possible increases usability by allowing a better match between gesture and action. The system uses computer vision requiring no sensors or markers on the user or background. The analysis of requirements for games, the architecture and implementation are discussed, as well as the results of several tests to evaluate how well each requirement is met.

Keywords: computer vision, gesture recognition, human-computer interaction, electronic games

Authors' contact:
{joao.bernardes, ricardo.nakamura}@poli.usp.br, tori@acm.org

1. Introduction

The possibility of relaying commands to a computer system using one's own hands and gestures has interested researches and users for a long time and was one of the first topics in user interface research, partly because it uses well-developed, everyday skills [Bowman 2005]. With the computational capacity available today and widespread use of image capture devices, even in domestic systems it is possible to implement this sort of interaction using computer vision. This brings the benefit of leaving the user's hands free of any gloves, cables or sensors. Gestures2Go, the system described here, provides this functionality and its implementation (in C++, illustrated in figure 1) is focused on electronic games.

Games are an ideal platform to test and popularize new user interface systems, for several reasons, such as an increased user willingness to explore in this medium [Starner et al. 2004]. There are many examples of academic research developing and studying new interfaces with games, particularly incorporating Augmented Reality [Bernardes et al., 2008]. The game industry has also introduced new (or of previously restricted use) interfaces and devices to the public. From the joystick to increasingly complex gamepads and controllers shaped as musical instruments, from datagloves to "pistols" that function as pointing devices and even haptic devices [Novint 2009], many are such examples, to the point that, today, some professionals are encouraged to play games to improve job-related skills [Dobnik 2004].

On the other hand, both the industry and academia acknowledge that new, more natural (and hopefully fun) interfaces are one way to attract new consumers to this economically important but still restricted market [Kane 2005]. And in the past few years, the search for these interfaces has been more widespread, continuous, well-publicized and commercially successful. After a popular gaming platform introduced motion and tilt detection in a simpler controller as its most innovating feature [AiLive 2007], motion detection was quickly added to other platforms and games and continues to be researched and improved upon. Several portable gaming systems, in particular, are taking advantage of motion and tilt sensing, touchscreens and even microphones in their interface. More recently still a project was unveiled to add interaction based on recognition of full-body motion, speech and faces to a popular platform [Snider 2009].

Despite this ebullience in game interfaces, the use of hand gestures, especially leaving the user's hands free, has seen little academic or commercial research in this area and is usually limited to analyzing only hand movement or a small number of hand postures. One of Gestures2Go's objectives is greater flexibility, to allow the use of a greater variety of gestures (currently defined by hand postures, movement or location and using both hands). Another important goal is that it must be easy to use for both players and developers. Gestures2Go should also be usable with existing games (designed for traditional interfaces) and allow multimodal interaction. These and other requirements arose, during system design, from an analysis focusing specifically on gestures and on game applications. Many of the same requirements exist in other applications as well, such as education or virtual and augmented reality, and the authors believe this system may be well suited for these applications, but will leave this discussion outside the scope of this paper.

2. Related Work

A few works have been proposed recently to use free hand gestures in games using computer vision. A multimodal multiplayer gaming system [Tse et al. 2007] combines a small number of postures, their location on a table-based interaction system and speech commands to interact with games and discusses results of using this platform to interact with two popular games. Interpreting movements or postures of the arms or the whole body is also usual. A body-driven multiplayer game system [Laakso & Laakso 2006] uses 8 postures of the two arms viewed from above, plus player location, to design and test the interaction in several games. Going further, tests with both functional prototypes and Wizard of Oz prototypes indicate that body movement patterns (such as running, swimming or flying), rather than specific gestures or trajectories, may be used to trigger similar actions on game characters [Hoysniemi et al. 2005].

Other tools facilitate the use of gesture recognition for applications in general, not only games. ICondensation [Isard & Blake 1998] is a probabilistic framework that allows the combination of different observation models, such as color and contours. HandVu [Kolsch et al. 2004] also uses condensation but provides a simpler interface to track hands in six predefined postures using skin color and a "flock" of Haar-like features. GART [Lyons et al. 2007] provides a high level interface to machine learning via Hidden Markov Models used to train and recognize gestures that consist only of movements (detected by sensors such as a camera, mouse or accelerometers). It is interesting to note that HandVu and GART can be combined to allow robust hand tracking and a larger number of gestures (combining postures and movement, like Gestures2Go) than either one isolated. Finally, EyesWeb [Camurri et al. 2003] is a framework with a graphical programming interface that presents several tools and metrics for segmentation and analysis of full body movements.

The literature regarding gesture recognition in general is vast and a complete review is beyond the scope of this paper, especially since established and comprehensive reviews [Pavlovic et al. 1997] as well as more recent but still comprehensive discussions [Imai et al. 2004] are available. Other works, when relevant to this implementation or future developments, are discussed in the correspondent sections.

3. HCI and Game-specific requisites

Both the use of gestures and having games as an application bring specific requirements to an interface and analyzing these requirements was one of the most important steps in designing Gestures2Go. For gesture-based interfaces, current research [Bowman et al. 2005, Shneidermann et al. 1998] point out the following:

Gestures are most often used to relay singular commands or actions to the system, instead of tasks that may require continuous control, such as navigation. Therefore, it is recommended that gestures be part of a multimodal interface [Bowman et al. 2005]. This also brings other advantages, such as decoupling different tasks in different interaction modalities, which may reduce the user's cognitive load. So, while gestures have been used for other interaction tasks in the past, including navigation [Mapes & Moshel 1995], Gestures2Go's primary requisite is to allow their use to issue commands. Issuing commands is a very important task in most games, usually accomplished by pressing buttons or keys. Often, games feature a limited number of commands, not even requiring all the buttons in a modern gamepad. Since other tasks, especially navigation, are very common as well, another requirement that naturally arises is that the system must allow multimodal interaction. Massively Multiplayer Online games (MMOs), in particular, often have much of their actual gameplay consisting of navigation plus the issuing of several commands in sequence [Fritsch et al. 2005].

Gesture-based interfaces are almost always "invisible" to the user, i.e. they contain no visual indicators of which commands may be issued at any particular time or context. To reduce short term memory load, therefore, the number of possible gestures in any given context, but not necessarily for the entire application, must be limited (typically to 7±2 [Miller 1956], or approximately 5 to 10 gestures). The gestures must also be highly learnable, chosen from the application domain so the gesture matches the intended command. Changing gears in a racing game, for instance, could be represented by pulling a fist towards or away from the user with the hand relatively low, as if driving a stick shift car, and pausing the game could be associated with an open palm extended forward, a well-known gesture meaning "stop". This means that while the system is not required to deal with a large

number of different gestures at any one time (which simplifies the implementation), being flexible by having a large number of possible gestures to choose from, so the interface designer may pick the most appropriate to associate with each user action, is indeed a requirement. Systems that violate either of these two requirements, requiring the memorization of a large number of gestures or limiting the space of possible gestures to only a few postures or movements, make the interface harder to learn and later to remember, reducing its usability.

The examples above (changing gears and stop) also show that the choice of each gesture for the interface depends not only on the application, context and command, but is also heavily culture-dependant, because the cognitive meaning of gestures may vary. In the case of gesture-based games, therefore, and with games being such a global market, localization could also entail changing which gesture is associated with each action [Bernal-Merino 2007]. All this leads to the requirement that the vocabulary of gestures in each context of the interface, while small, must be as simply and quickly modifiable as possible. Systems that require retraining for each set of possible gestures, for instance, could prove problematic in this case, unless such training could be easily automated.

The interface should also accept small variations for each gesture. Demanding that postures and movements be precise, while possibly making the recognition task easier, makes the interaction considerably harder to use and learn, demanding not only that the user remember the gestures and their meanings but also train how to do them precisely, greatly reducing usability.

It could be argued that, for particular games, reducing the usability could actually be part of the challenge presented to the player (the challenge could be remembering a large number of gestures, or learning how to execute them precisely, for instance). While the discussion of whether that is a good game design practice or not is beyond the scope of this paper, Gestures2Go opts for the more general goal of increasing usability as much as possible. This agrees with the principle that, for home and entertainment applications, ease of learning, reducing user errors, satisfaction and low cost are among the most important design goals [Shneidermann et al. 1998].

The system should also allow playing at home with minimal setup time required. Players prefer games where they can be introduced to the action as soon as possible, even while still learning the game and the interface [Hong 2008]. Therefore, the system should not require specific background or lighting conditions, complex calibration or repeated training. Allowing the use of the gesture-based interface with conventional games is also advantageous to the user, providing new options to enjoy a larger number of games. From the developer point of view, the system should be as easy as possible to integrate within a game, without requiring specific knowledge of areas such as computer vision or machine learning.

Finally, processing and response times are important requirements. Despite the growing availability of multi-core gaming platforms, it is still desirable that gesture recognition processing time be as low as possible, freeing processing power to other tasks such as artificial intelligence and physical simulation. It is limited by the acceptable response time, which, in turn, depends on the game. Performing a gesture, for instance, will almost always be slower than pressing a button or key, so this sort of interface is probably not a good choice for reflex-based games such as first person shooters. A genre that has already been mentioned as a good match for this sort of interface is MMOs. Not only much of their gameplay consists of navigation and issuing commands, MMOs use several strategies to deal with network latency [Fritsch et al. 2005] that also result in not penalizing the slower input from gestures, when compared, for instance, with button pressing. Such strategies include reducing the number of commands necessary in a fixed amount of time (for instance, it is common to "enter or exit attack mode", instead of repeating a command for each attack) and accepting the queuing of only one new command while the action triggered by the last one has not finished (and actions are set up to take some time, usually spent with animations or special graphical effects). In the game Everquest 2, for instance, Fritsch et al. report that the use of these strategies, with actions usually taking 1000ms, makes the game playable with latencies of up to 1250ms. A more practical bound, however, pointed after the analysis of several related works, is around 250ms for interactive games [Henderson & Bhatti 2003]. In a setup such as the one described above, that would leave one second to be divided between gesture performance and system response time and this is the parameter that will be used for Gestures2Go. This applies, of course, even for games designed for regular interfaces. When designing a game specifically to explore gestures, similar game design strategies or even new ones could be adopted to compensate for the time the user spends performing the gesture.

4. Gestures2Go

Because one of the requirements for this system was ease of use, both for the player and the developer, it was named Gestures2Go to imply that the gesture recognition is ready to go, to take home, with little extra work. It consists of an abstract framework that divides the system in modules and defines the interface between these modules and, currently, of a single, simple implementation of this framework. It is important to note that the requirements discussed in section 3 apply to the current implementation, which is focused on games, and not to the abstract framework.

The computational task of identifying a gesture from a known vocabulary of possibilities is often divided in gesture modeling, analysis and recognition [Pavlovic et al. 1997].

Gesture modeling consists in how a gesture is defined by the system, from a computational point of view (since definitions of gesture abound in other areas). Gesture2Go's abstract framework defines a gesture as an initial hand posture, an optional movement of the entire hand through an arbitrary path and a final posture, which is optional if the movement is omitted but mandatory otherwise. The starting location of the hand, relative to the user's head (left or right, above, below or roughly aligned with the head), is also an optional parameter of this definition, since it often changes the meaning of a gesture. This means that a gesture may consist of a single posture, of an initial and a final posture or of an initial posture, a movement and a final posture, all depending or not on the initial hand position. It also means that changes of posture during the movement are not taken in consideration, since these changes rarely have semantic meaning [Quek 1994]. While the abstract framework also includes variable parameters in the gesture definition (such as speed or pointing direction), the simple implementation described here does not deal with parametric gestures. Finally, the abstract framework does not specify how each part of the gesture definition is actually modeled (each is identified by a string or numerical ID), so it can vary in each implementation. The hand posture could, for instance, be modeled as a collection of values for the degrees of freedom of a particular hand model, or it could consist of a list of 2D or 3D points of the hand's contour.

During the analysis phase, the gesture's spatial and temporal parameters (which depend on each model) are obtained from sensor data (in this case, from an image or a set of images) and this data is used during the recognition phase to identify the gesture within the vocabulary of possibilities. Analysis and recognition are often, but not necessarily, tightly inter-related.

4.1 The Abstract Framework

Figure 2 shows a UML Activity Diagram representing Gesture2Go's object flow model.

G2gGesture is responsible for the gesture model, while *G2gAnalysis* and *G2gRecognition* define the interfaces for the classes that will implement gesture analysis and recognition. To these activities are added image capture and segmentation. *G2gCapture* provides an interface for capturing 2D images from one or multiple cameras or pre-recorded video streams (mostly for testing). The images must have the same size, but not necessarily the same color depth. A device could provide, for instance, one or more color images and a grayscale image to represent a dense depth map. *G2gSegmentation* should usually find in the original image(s) one or both hands and possibly the head (to determine relative hand position).

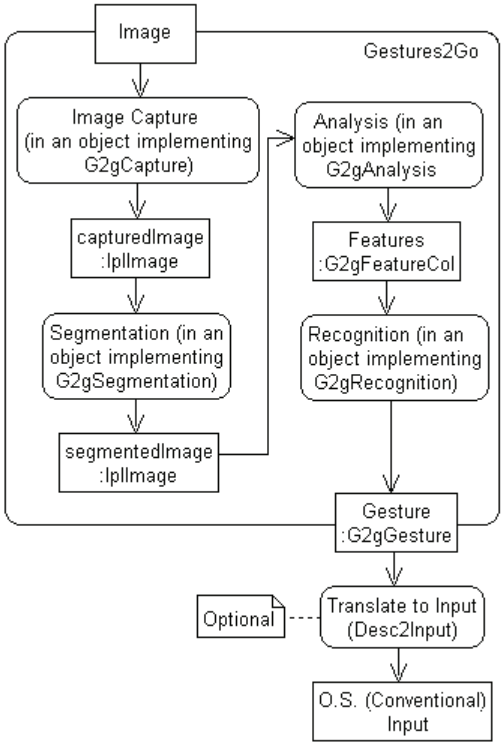

Figure 2: Gesture2Go's Object Flow Model

Figure 2 shows that the usual flow of information in Gestures2Go in each time step is as follows: one or more images serve as input to the image capture model, which makes these images available as an OpenCV's *IplImage* object [OpenCV 2009]. The segmentation uses this image and provides a segmented image as an object of the same class (and same image size, but not necessarily color depth). Based on the segmented image, the analysis provides a collection of features as a *G2gFeatureCol* object which are in turn used by the recognition to output a gesture.

G2gFeatureCol is a collection of *G2gFeature* objects. *G2gFeature* contains a identifier string to describe the feature and either a scalar and an array of values (more often used) or an image (useful, for instance, for features in the frequency domain). *G2gFeature* already defines several identifiers, for those features most often found in the gesture recognition literature, to facilitate the interface between analysis and recognition, but user-created identifiers may also be used.

Desc2Input is an optional module that accompanies but is actually separate from Gestures2Go. It is responsible for facilitating, in a very simple way, both multimodal input and integration with games or engines not necessarily aware of Gesture2Go. It simply translates its input, which is a description (a numerical or string ID or a XML description, for instance) that may be supplied either by Gestures2Go or any other system (and here lies the possibility of multimodal interaction), into another type of input, such as a

system input (like a key down event) or input data to a particular game engine. In one of the tests, for instance, gestures are used for commands and a dancing mat is used for navigation.

Because this architecture consists mostly of interfaces, it is possible to create a single class that, through multiple inheritance, implements the entire system functionality. This is usually considered a bad practice in object orientation (should be avoided) and is actually one of the reasons why aggregation is preferred to inheritance [Eckel 2003]. There are design patterns that could have been used to force the use of aggregation and avoid multiple inheritance, but Gestures2Go opts for allowing it for a reason. Gesture recognition may be a very costly task in terms of processing, and must be done in real time for the purpose of interaction. Many algorithms may be better optimized for speed when performing more than one task (such as segmentation and analysis) together. Furthermore, analysis and recognition are very tightly coupled in some algorithms and forcing their separation could be difficult. So, while it is usually recommended to avoid using multiple inheritance and to implement each task in a different class, making it much easier to exchange one module for the other or to develop modules in parallel and in teams, the option to do otherwise exists, and for good reason.

Finally, all Gestures2Go classes must implement *init( )* and *cleanup( )* methods which are preferred to using the *new* and *delete* operators (the system is implemented in C++) to avoid problems with multiple inheritance and with synchronization.

4.2 Implementation

The requirement analysis pointed that an implementation of the abstract framework described above specifically for games should have the following characteristics: minimum need for setup, low processing demand even though the response time may be relatively high, a high number of possible gestures but with only a small and easily modifiable vocabulary in any one context, tolerance to variations in the execution of gestures, allow multimodal interaction and make development of games using gestures as easy as possible. With these requirements in mind and assuming that a single player in the scene will interact through gestures, this implementation attempts to find the simplest solution for each of the activities shown in figure 2.

Segmentation is based on skin color, to find both hands and the head. A *G2gSimpleSkinSeg2* (a class which implements *G2gSegmentation*) object performs a simple threshold operation on the captured image, in the HSV color space, taking in account both hue and saturation. For most people, skin color lie in a small interval between the red and yellow hues, due to blood and melanin, respectively [Fleck & Forsyth 2009], so using hue is a good way to identify a large range of lighter or darker skin tones, even in different illumination conditions. Saturation is used mostly to remove regions that are either too light or too dark and may end up showing a hue similar to the skin.

At first, fixed average values and tolerances were adopted for the skin's hue and saturation. Testing in different lighting conditions, environments and using different cameras, however, showed large variations for these values in the captured images, either due to different lighting conditions or differences in the white balance [Viggiano 2004] performed automatically by the cameras (and, in most cases, with no "off" option). *G2gSimpleSkinSeg2* was then incremented with methods to accumulate and calculate averages and standard deviations for hue and saturation of several, arbitrary rectangular skin-colored regions. This allows an application to add a quick calibration step so the segmentation may use adequate skin hue and saturation values for the threshold operation.

Finally, after tests in an environment where the background actually has a hue very similar to the skin's, a fixed background removal operation was added as an option. Figure 1 shows a sample result of this operation. Even with a color tolerance of 50 in a 256x256x256 RGB space, about half of the pixels do not match the recorded background (not showing as black), even when this background is far enough that its actual appearance is unlikely to change due to the presence of the hand. This problem is minimized by applying a 3x3 erosion operation after the background removal, also illustrated in figure 1, but due to local corrections imposed by the camera a region around the foreground elements still shows, looking like an "aura" around the color segmented hand images in figure 1.

The system, currently, does not segment the arm from the hand, which imposes the limitation that users must wear long sleeves. This is considered a serious limitation. Even without any information about hand posture, for most of them the arm could be segmented by finding the direction of its major axis, finding the point of minimum width or abrupt change in direction along this axis (the wrist) and segmenting there [Yoon et al. 2006]. This does not work well if only a small length of arm is showing, however, or for certain postures (such as preparing a "karate chop").

Other segmentation strategies that do not require knowledge of the hand's posture were attempted, such as using color histograms and probabilities instead of the simple average and deviation, as well as the use contour information, but so far showed little improvement and more computational cost.

The first step of the analysis activity, implemented in the *G2gSCMAnalysis* class, is to find the connected components in the segmented image. The system does not assume that the background is fixed or that there are no other skin colored regions in the image, but it does presume that the player using gestures is the

closest person to the camera, so it can assume that the three largest connected components correspond to the user's hands and face. There is also a minimum number of pixels for a connected component to be accepted as a region of interest. If only 2 components above this minimum size are found, the system assumes that the missing component corresponds to the user's non-dominant hand and if only one is present, it is assumed to be the head (the head was cropped from figures 1 and 3). To further simplify the identification of the hands and head, this implementation assumes that the left hand is the leftmost region with the head in the middle and the right hand to the right. While this certainly limits user movements and the number of possible gestures, it was considered a valid limitation in this case and, during informal testing, was accepted with no complaint from the users, who obeyed it most of the time even when not informed of it. This first step also reduces noise left after the segmentation and eliminates from the analysis other people who might wander in the background.

Analysis and recognition of the gestures themselves adopt a divide and conquer strategy [Wu & Huang 1999], separating the recognition of hand posture and hand movements. Postures are recognized through estimation by synthesis (ES), i.e. the real hand's image is compared with images synthesized from a 3D hand model so that 3D posture information (the parameters used to model the hand's posture) is obtained comparing only 2D images, instead of trying to match a 3D model to the real image, which can be accurate but computationally expensive and complicated by the presence of postures with self occlusion [Imai et al. 2004]. Unlike most applications of ES methods, however, here it is not necessary to determine hand posture continuously and differentiate between postures with only small differences. Because tolerance of variation in postures is one of the system's requirements, it is not only acceptable but necessary that small differences in posture be disregarded. This implementation, therefore, may sidestep one of the most serious complication of ES methods. It only needs to compare the real hand image with a small number of possible postures, instead of thousands of possibilities. When no acceptable match is found, the system simply assumes the user is not performing a command gesture.

As in other ES methods [Shimada et al. 2001, Imai et al. 2004], the features *G2gSCMAnalysis* provides are based on the hand's 2D contour. The most important feature is a vector of the distances between the hand's centroid and a fixed number of points on the contour. These points are shown in figure 1. This vector is normalized in the analysis, so the maximum distance always corresponds to the same value and the features are scale-invariant, reducing the influence of the distance between the hand and the camera. All features for the vocabulary of possible, modeled postures are pre-calculated so only those for the real hand need to be determined in each execution step. Currently the number of points sampled from the contour in the feature vectors is, somewhat arbitrarily, set at 128. This number has shown to be small enough to allow fast computation and large enough that it is not necessary to worry about choosing points near remarkable contour features (usually local maxima and minima corresponding to tips and bases of fingers).

G2gSCMRecognition implements both posture and movement recognition. Posture recognition consists simply of comparing the feature vector obtained from the real hand's captured image with each vector for all the possible postures and finding the posture that minimizes the mean squared error between these two vectors. If the minimum error is still larger than a tolerance value, no posture is recognized (recognition returns a "not found" constant).

Unlike other ES implementations, however, the observed vector is not made rotation-invariant during recognition (by rotating it during each comparison so extremal points coincide with the model). While some tolerance in posture recognition is desired, rotation-invariance is not. Should this operation prove necessary, to avoid incorrect results due to the accumulation of many small errors caused by a small rotation, it could still be implemented while leaving the algorithm sensitive to rotation because recognition uses yet another feature: the angle between the highest point in the contour and the centroid. This feature, also provided by *G2gSCMAnalysis*, is currently used to speed up recognition by discarding, before the calculation of the mean squared error, any posture with an angle that differs by more than a certain tolerance from the one in the observed image. The highest point (usually a fingertip) is easy to determine because the contour-finding algorithm is implemented in a way to always find this point first. This angle could also be used to account for hand rotation if the vector of distances was made rotation-invariant, but tests so far have not shown the need for this operation.

The analysis also provides the centroid's absolute location in the image and its area (or number of pixels), which are used for movement recognition. Only 12 movements are recognized: left, right, up, down, back, forward, 4 diagonals, clockwise and counter-clockwise approximate rotations. The movement is temporally segmented by the gesture's initial and final postures, so it can be identified as one of these possibilities by a simple set of conditions, similar to a two stage scheme described in the literature [Mammen et al. 2001]. For the back and forward movements, the initial and final posture of the hand must be the same, since this movement is estimated by the variation in area.

In the current implementation, a gesture may be defined by movements and initial relative locations of both hands, but only postures of the dominant one (currently the right hand, but the next version will allow choosing left or right) are identified. There are now 41 postures available. Adding more postures is

quite simple and others were considered and could have been added, but they were either meaningless, quite hard to perform or had the same contour in a 2D image. With this number of available postures and movements, and remembering that a gesture might consist of one or two postures, or a movement bound by two postures that may be different (except when moving back or forward), there are almost 20,000 available gestures for the dominant hand alone, even before considering its location relative to the head or the movement of the other hand.

Finally, *Desc2Input*'s implementation in the current version, for MS Windows only, has only two public methods: *associate* and *sendDesc*. The *associate* method receives a description (a string, representing a gesture or any other event, such as stepping on a dancing mat's "button") and the system input (key press, mouse move or click) and parameters (such as key or position) associated to that description. The *sendDesc* method only receives a description and indicates that *Desc2Input* must generate the associated input (which is broadcast to all windows). A priority for future versions is making this module easier to use, adding alternatives that require little programming (leaving the association of gestures and commands to an external configuration file, for instance).

5. Tests and Results

Four prototype applications were created to test the system in different conditions. The first priority was to verify the posture analysis and recognition strategy, independent of segmentation. To accomplish that, 120 already segmented pictures of hands in different postures were stored and ran through the analysis and recognition modules. These images were segmented using the same algorithm described before but were chosen manually at moments when it worked adequately (as in the examples shown in figure 3).

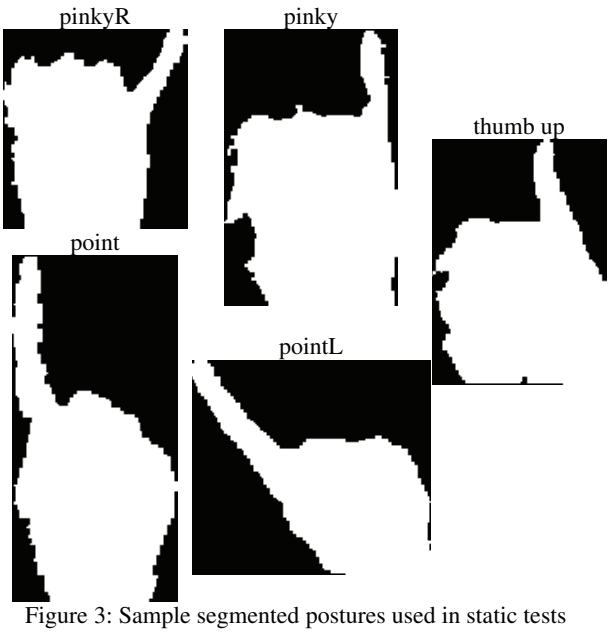

Figure 3: Sample segmented postures used in static tests

To allow the comparison of every posture with every other one, the angle difference between the highest point in each posture was discarded and the mean square error between the distance vectors was recorded. Table 1 shows the results, truncated to the nearest decimal, of one such test, comparing 15 postures. More postures are not shown due to the limited space. This particular test was chosen specifically because it contains similar postures that show problematic results.

In all cases the correct posture was identified (i.e. had the minimum error), as shown by the values with a gray background in table 1. In 8 cases, however, incorrect postures showed a low error as well (shown in bold on white). The system considers error values below 1 as possible matches. So, if "pinkyR" had not been one of the possible postures, for instance, "pinky" would have been accepted by the system as "pinkyR". Figure 3 shows these problematic postures. Two of these cases (pinky and pinkyR, point and pointL) are postures where a single finger is raised and that differ from each other by this finger's angle. Using the angle of the highest point as a feature eliminates these incorrect matches. The other mismatch that might have occurred is between the postures with the pinky up and the thumb up posture, but as seen in figure 3, these postures are actually quite similar. In all these static tests, all postures were recognized correctly but a few similar ones showed possible mismatches. In the test illustrated by table 1, for instance, only 8 comparisons in 225 were possible mismatches, approximately 3.5%.

Table 1: Sample static posture comparison

two	tu	three	pt	pointL	point	pinky	pinkyR	palm	open	openL	hm	hl	fist	claw	
15.9	9.7	13.3	13.5	15.6	16.1	11.1	9.2	9.4	7.4	5.5	14.1	9.7	5.7	**0.8**	claw
14.5	10.2	10.4	16.1	18.0	19.2	11.3	10.3	8.6	9.8	10.1	16.2	12.1	**0.8**	4.3	fist
8.6	2.3	7.3	3.6	3.3	2.9	3.4	3.4	3.2	6.6	9.8	3.3	**0.9**	13.6	12.5	hl
4.1	1.3	9.9	1.4	2.8	3.0	1.8	2.6	4.0	9.3	10.1	**0.4**	2.2	16.8	17.3	hm
10.4	6.4	10.3	8.8	11.4	11.9	7.0	6.0	9.4	3.4	**0.3**	9.8	7.7	11.5	6.1	openL
12.2	5.2	8.5	9.1	9.5	10.0	6.7	6.6	8.6	**0.4**	2.2	8.1	5.6	12.6	8.6	open
7.2	2.6	9.0	4.0	3.8	4.4	2.6	3.3	**0.1**	8.7	9.8	4.6	3.9	7.8	10.5	palm
2.6	**0.4**	5.2	1.7	3.0	3.4	**0.3**	**0.5**	2.5	6.4	7.0	1.8	2.1	10.6	11.1	pinkyR
3.8	**0.5**	5.3	2.3	2.6	3.0	**0.3**	**0.7**	1.9	6.6	7.9	2.2	2.1	10.9	11.8	pinky
7.5	2.8	9.4	1.8	**0.3**	**0.1**	2.3	3.4	3.8	9.7	11.2	2.8	2.9	18.8	17.3	point
7.8	3.0	9.5	2.0	**0.1**	**0.2**	2.5	3.8	3.6	10.1	11.9	3.1	3.4	18.5	17.6	pointL
2.8	1.5	8.2	**0.2**	2.2	2.3	1.4	1.3	3.4	9.6	9.1	1.8	2.7	15.2	14.7	pt
7.1	4.9	**0.4**	7.4	7.9	8.4	5.2	5.3	8.0	7.5	9.7	9.1	7.3	13.7	15.2	three
3.8	**0.2**	5.0	2.3	3.1	3.4	**0.6**	**0.9**	2.5	5.4	7.2	1.6	1.4	10.6	11.1	tu
0.6	3.8	9.4	2.9	8.3	8.1	4.0	3.0	8.5	12.7	11.0	4.5	5.8	17.7	18.0	two

A second test application shows identified postures in real time and allows the verification of the effects of the segmentation. It requires a few seconds for setup,

showing a region on the screen that the user must "cover" with a region of skin so initial averages and deviations for skin color can be determined. While the application allows this to be done several times (to capture, for instance, the colors of the palm and back of the hand as well as face regions), showing either many or any single region of skin have always given similar results during tests. The application also includes the options of recording and removing a known background and can either show a color image of the foreground or a monochrome image of the segmented skin regions. While showing the monochrome image, if a posture is identified the application also displays its description on the bottom of the screen. This application also identifies and displays the 8 possible movements. Actually, a gesture was defined for each movement, all 8 having as both initial and final posture a closed fist (which is very accurately identified by the system). The images labeled as "Erosion" and "Posture" in figure 1 are actually regions from screenshots of this application.

During the tests with this application, analysis and recognition continued to perform well when the images were well segmented. Often, however, a finger, usually the thumb or pinky, would disappear from the segmented image or only parts of the fingers would show, leading to postures not being recognized or for mismatches (such as an open palm identified as mimicking a claw). This was mostly due to problems with the illumination and image capture, such as a bloom showing between the fingers if the open hand was in front of a light source or bright light sources reflecting specularly from large regions of skin. Both make large skin regions show as white. Even in these environments with no controlled (and problematic) illumination, the system identified the right posture most of the time. Another problem that occurred during these tests happened when the long sleeves worn by the subjects slid down the wrist, showing a portion of the forearm. Only 2 or 3 centimeters needed to show to cause a dramatic drop in the recognition's quality. During these tests, the movements were always recognized correctly.

While Gestures2Go should be primarily used to issue commands with gestures, a third application was built to evaluate its use to select objects, replacing the use of the mouse. A posture was associated with moving the mouse and relative changes in hand position while that posture was recognized were mapped to relative positions in the mouse pointer using *Desc2Input*. Two other postures were associated with left and right clicks. The hand moved only in a small region of a 640x480 image while the mouse should move over a 1024x768 region, so the linear mapping between movements increased the hand's vertical and horizontal movements by different constants to apply it to the mouse. The system was still relatively easy to use even to click on smaller objects on the screen.

Finally, postures, movements, using the hand to move the mouse pointer and click and the use of a dancing mat for navigation were put together in a fourth test application which was used to control a popular MMO. Using the hand to move the mouse pointer and clicking was only necessary to manipulate some objects in the scenery. A gesture was associated with the command to select the next possible target and several gestures were associated with different actions to be performed on this target. This interface was especially adequate to this particular MMO because most actions are accompanied by easily identifiable hand motions of the player's avatar, so the mapping between gesture and game action was natural, very visible and enjoyable. To navigate in the game world using the dancing mat, it was connected to the computer's parallel port and a class was created to read its inputs and send them to Desc2Input to be translated as the arrow keys and commands for actions such as jumping. Because in systems derived from Windows NT only applications running in kernel mode can access the parallel port, it was necessary to either write a device driver or use an existing one. Using Inpout32 [logix4u 2009] was the chosen solution. It is a DLL with an embedded driver and functions for reading and writing to the parallel port (inp32 and out32). Up to the time of this writing, unfortunately, permission to use this MMO's name and images had not yet been granted by the publisher.

The performance of each module was also tested, using a 3GHz Intel Core 2 Duo CPU and 2GB of RAM (the test process ran in only one core, however). Table 2 shows approximate average times measured for each task in 375 tests (5 tests of 5s at 15 frames per second).

Table 2: Performance

Activity		**Time (ms)**
Segmentation		13.600
Analysis	Components	0.650
	Moments	0.013
	Features	0.003
Recognition	10 Postures	0.002
	Movement	<0.001

Table 2 shows how segmentation is by far the most costly activity. During analysis, finding the connected components is also the most time consuming task, but still only takes less than a millisecond. Finding the image moments for one hand's connected component takes approximately 13µs only because OpenCV's function calculates up to third order moments, while the system only requires moments of orders 0 and 1, so this operation could be easily sped up, but it is clearly not a priority. Calculating all features needed for recognition and the recognition itself were extremely fast during these tests, at less than 5µs. That's assuming there are 10 possible postures (recognition time increases linearly with possible postures) and a worst case scenario where the angle difference is never above tolerance, so the mean square error between distance vectors is calculated for every possibility. Movement

recognition consists of only a few conditions and happened too fast to get accurate measurements. With these results, the system satisfies the requirement of low processing demand and should it be necessary to make it faster, it is trivial to parallelize the segmentation, either to run in more cores or to be done in the GPU. These processing times, however, indicate that finding a more robust segmentation strategy is much more important than increasing its performance.

6. Conclusion

This current implementation of Gestures2Go, focused specifically on games and other similar applications, satisfies most of the requirements for gesture-based interfaces and games which were studied during the system's design phase.

While there is need of some setup, to record the background and calculate the player's skin color parameters, this setup only takes a few seconds. Each execution step takes less than 15ms in a single 3GHz core, satisfying the requirements for low processing demand, especially considering that in most contexts the system must only differentiate between 5 to 10 gestures. However, combining 41 (or more) postures of one hand and 12 movements and initial hand locations (relative to the head) for both hands creates a vocabulary of thousands of possible gestures, greatly increasing the chance that the interface designer can find an appropriate gesture to associate with an action. Desc2Input facilitates multimodal interaction and the system as a whole is quite tolerant to variations in gesture execution, both for postures and movements.

One requirement cannot be considered satisfied yet, however: simplifying the development of games with gestures. *Desc2Input* should be responsible for this requirement, but currently its interface only allows the association of descriptions and inputs by hard coding them using the associate function. Furthermore, its current version is provided as source code that must be included within the same project as the gesture recognition system and systems for interpreting other modes of interaction (such as the dancing mat used in one of the tests, or speech recognition). This makes the system's use by programmers much more complex than desired. It is a priority for future works, therefore, to develop a better interface for *Desc2Input*. The next system's version will allow the association of descriptions and inputs though an external xml configuration file and *Desc2Input* will be available not only as source code but as a DLL to include in projects as well as a standalone executable that receives descriptions via sockets from different modules responsible for complementary interaction modes. Gestures2Go will also include a standalone application that generates regular system inputs from command gestures so that this sort of interface may be used with any other interactive application simply customizing a configuration file associating gestures to inputs, without requiring a single line of programming. Another standalone application is in development to facilitate this configuration: instead of editing the configuration file directly, the user simply shows initial and final posture to the system and selects, in a graphical interface, movements, locations and which input that gesture must generate. A final improvement in this area is the integration of Gestures2Go with a game engine, but this depends on the engine's architecture and is beyond this paper's scope.

Another priority for future works is improving the segmentation. One of the system's requirements is that it must not demand controlled or special lighting or unusual or expensive equipment and, under those severe limitations, the segmentation actually works considerably well. But it is still the less robust part of the system and causes frequent and noticeable errors under some lighting conditions. Several robust probabilistic solutions exist to track hands and their contours, such as using variations of the condensation algorithm [Isard & Blake 1998]. Most of these solutions require knowledge either of one fixed hand posture, or a small number of postures and a transition model between them [Liu & Jia 2004] which complicates the addition of new postures and gestures. Even these methods often use depth data to aid in segmentation. Other methods do not require a known model for the hand but only track its position, not the contour, which is necessary for Gestures2Go. One promising approach that will be tested as soon as possible within this system is tracking the hand and its contour with no hand model information by using Kalman filters to estimate both the hand's movement and the positions of control points of curves that define hand shape [de Bem & Costa 2006]. This strategy will be adopted if tests show that its performance and accuracy are adequate while tracking enough control points to model a rapidly changing hand contour.

Using depth data [Nakamura & Tori 2008] is another planned improvement to the system, both to the segmentation and to allow a greater number of postures, such as pointing postures. Lastly, formal usability tests must be conducted to determine whether the interaction techniques using Gestures2Go in a MMO are effective in the context of games.

References

AILIVE, 2007. LiveMove White Paper. Available from: http: //www.ikuni.com/papers/LiveMoveWhitePaper_en.pdf [Accessed 24 July 2009].

BERNARDES, J. ET AL, 2008. Augmented Reality Games In: *Extending Experiences: Structure, analysis and design of computer game player experience*. Lapland University Press, p. 228-246.

BERNAL-MERINO, M., 2007. Localization and the Cultural Concept of Play. Available from: http://www. gamecareerguide.com/features/454/localization_and_the_ cultural_.php [Accessed 24 July 2009].

BOWMAN, 2005. 3D User Interfaces: Theory and Practice. Addison-Wesley.

CAMURRI ET AL., 2003. Analysis of expressive gestures in human movement: the EyesWeb expressive gesture processing library. In: *Proc. XIV Colloquium on Musical Informatics*.

DE BEM, R., COSTA, A., 2006. Rastreamento de visual de múltiplos objetos utilizando uma abordagem livre de modelo. In: *Proc. XVI Congresso Brasileiro de Automática*, 2760-2765.

DOBNIK, V., 2004. Surgeons may err less by playing video games. Available from: http://www.msnbc.msn.com/id/ 4685909 [Accessed 24 July 2009].

ECKEL, B., 2003. Thinking in C++. Prentice Hall.

FLECK, M. & FORSYTH , D., 2009. Naked people Skin Filter. Available from: http://www.cs.hmc.edu/ ~fleck/naked-skin.html [Accessed 24 July 2009].

FRITSCH ET AL., 2005. The Effect of Latency and Network Limitations on MMORPGs (A Field Study of Everquest 2). In: *Proc. of NetGames '05*.

HENDERSON & BHATTI, 2003. Networked Games – a QoS-Sensitive Application for QoS-Insensitive Users? In: *Proc. ACM SIGCOMM 2003 Workshops*.

HONG, T., 2008. Shoot to Thrill. In: *Game Developer* 15(9) p. 21-28.

HOYSNIEMI ET AL., 2005. Children's Intuitive Gestures in Vision-Based Action Games. In: *Communications of the ACM* 48(1), p.44-50.

IMAI, A. ET AL., 2004. 3-D Hand Posture Recognition by Training Contour Variation. In: *Proc. Automatic Face and Gesture Recognition 2004*, p. 895-900.

ISARD, M., BLAKE, A., 1998. ICondensation: Unifying low-level and high-level tracking in a stochastic framework. In: *Proc. 5th European Conf. Computer Vision*.

KANE, B., 2005. Beyond the Gamepad panel session. Available from: http://www.gamasutra.com/features/ 20050819/kane_01.shtml [Accessed 24 July 2009].

KOLSCH, ET AL., 2004. Vision-based Interfaces for Mobility. In: *Proc. Intl. Conf. on Mobile and Ubiquitous Systems*.

LAAKSO, S., LAAKSO, M., 2006. Design of a Body-Driven Multiplayer Game System. In: *ACM CCIE* 4(4).

LIU, Y., JIA, Y., 2004. A Robust Hand Tracking and Gesture Recognition Method for Wearable Visual Interfaces and its Applications. In: *Proc. ICIG '04*.

LOGIX4U, 2009. Inpout32.dll for Windows 98/2000/NT/XP. Available from: http://logix4u.net/Legacy_Ports/Parallel_ ort/npout32.dll_for_Windows_98/2000/NT/XP.html Accessed 24 July 2009].

LYONS, H. ET AL., 2007. Gart: The gesture and activity recognition toolkit. In *Proc. HCI International 2007*.

MAMMEN, J.; CHAUDHURI, S. & AGARWAL, T. Simultaneous Tracking Of Both Hands By Estimation Of Erroneous Observations. In: *Proc. British Machine Vision Conference* 2001.

MAPES, D., MOSHEL, J.,1995. A Two Handed Interface for Object Manipulation in Virtual Environments. In: *Presence: Teleoperators and Virtual Environments* 4(4), p. 403-416.

MILLER, G., 1956. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. In: *The Psychological Review* 63, p. 81-97.

NAKAMURA, R., TORI, R., Improving Collision Detection for Real-Time Video Avatar Interaction. In: *Proc. X Symp. on Virtual and Augmented Reality*, p. 105-114.

NOVINT, 2009. Novint Falcon. Available from: http://home.novint.com/products/novint_falcon.php [Accessed 24 July 2009].

OPENCV, 2009. Available from: http://sourceforge.net/ projects/opencvlibrary/ [Accessed 24 July 2009].

PAVLOVIC ET AL., 1997. Visual Interpretation of Hand Gestures for Human Computer Interaction: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19(7), p. 677-695.

QUEK, 1994. Towards a vision-based hand gesture interface. In: *Proc. Virtual Reality Software and Technology Conference 1994*.

SHIMADA ET AL., 2001. Real-time 3-D Hand Posture Estimation based on 2-D Appearance Retrieval Using Monocular Camera. In: *Proc. Int. Workshop on RATFG-RTS*, p. 23–30.

SHNEIDERMANN, 1998. Designing the user interface: strategies for effective human-computer interaction. Addison Wesley, 3. ed.

SNIDER, M., 2009. Microsoft unveils hands-free gaming. In: *USA Today*, 1 June 2009. Available from: http://www. usatoday.com/tech/gaming/2009-06-01-hands-free-microsoft_N.htm [Accessed 24 July 2009].

STARNER ET AL., 2004. Mind-Warping. In: *Proc. ACM SIGCHI Advances in Computer Enternatainment 2004*, p. 256-259.

TSE ET AL., 2007. Multimodal Multiplayer Tabletop Gaming. In: *ACM CIE* 5(2).

VIGGIANO, J., 2004. Comparison of the accuracy of different white balancing options as quantified by their color constancy. In: *Proc. of the SPIE* 5301, p. 323-333.

WU, Y., HUANG, T., 1999. Capturing Articulated Human Hand Motion: A Divide-and-Conquer Approach. In: *Proc. IEEE Int'l Conf. Computer Vision*, p. 606-611.

YOON, T. ET AL., 2006. Image Segmentation of Human Forearms in Infrared Image. In: *Proc. 28 IEEE EMBS Annual International Conf.*, p. 2762-2765.