Real-time skeleton tracking for embedded systems

Foti Coleca[ab], Sascha Klement[b], Thomas Martinetz[a] and Erhardt Barth[a]

[a]Institute for Neuro- and Bioinformatics, University of Lübeck, Ratzeburger Allee 160, 23562 Lübeck, Germany

[b]gestigon GmbH, Maria-Goeppert Straße 23562 Lübeck, Germany - www.gestigon.com

ABSTRACT

Touch-free gesture technology is beginning to become more popular with consumers and may have a significant future impact on interfaces for digital photography. However, almost every commercial software framework for gesture and pose detection is aimed at either desktop PCs or high-powered GPUs, making mobile implementations for gesture recognition an attractive area for research and development. In this paper we present an algorithm for hand skeleton tracking and gesture recognition that runs on an ARM-based platform (Pandaboard ES, OMAP 4460 architecture). The algorithm uses self-organizing maps to fit a given topology (skeleton) into a 3D point cloud. This is a novel way of approaching the problem of pose recognition as it does not employ complex optimization techniques or data-based learning. After an initial background segmentation step, the algorithm is ran in parallel with heuristics, which detect and correct artifacts arising from insufficient or erroneous input data. We then optimize the algorithm for the ARM platform using fixed-point computation and the NEON SIMD architecture the OMAP4460 provides. We tested the algorithm with two different depth-sensing devices (Microsoft Kinect, PMD Camboard). For both input devices we were able to accurately track the skeleton at the native framerate of the cameras.

Keywords: Self-organizing maps, pose estimation, gesture recognition, time-of-flight

1. INTRODUCTION

Although hands-free gesture control is not a new technology, it has seen little use in mobile computing. Almost every commercial software framework on the market today is aimed at desktop PCs[1 2 3] , due to the computing power required by the algorithms they use. This may well be because of the daunting task of pose estimation, especially the estimation of the human hand pose. From a computer vision point of view, it is a very complex object, with a high number of degrees of freedom, which makes it difficult to detect and track, more so in real time.

As mobile platforms shrink in size, gestural interfaces will start to become a viable alternative to touch-based interaction - the user can do only so much with a limited amount of touchscreen real estate, before his fingers start to get in the way. As a practical application in the area of computational photography, gesture control could be a potent interface from which the user could input parameters such as regions of interest, focus, depth of field, effects etc. In other embedded applications, being able to control appliances such as TVs from a distance without the need for a remote control is just starting to become a feature in the new range of consumer devices, although only for high-end models (such as Samsung's *Smart* TVs), due to the limitations described above.

In this paper we use a novel approach to pose estimation based on self-organizing maps (SOM). It has been introduced in[4] for tracking the upper body and is here extended to full-body and hand pose. The SOM-based algorithm has the advantage of needing few resources, and is ideal for implementation on an embedded platform. The core algorithm is simple but powerful enough to be able to fit a skeleton to a point cloud of data in a variety

Further author information: (Send correspondence to E.B.)

F.C.: E-mail: coleca@inb.uni-luebeck.de

S.K.: E-mail: sascha.klement@gestigon.de

E.B.: E-mail: barth@inb.uni-luebeck.de, Telephone: +49 451 5005503

T.M.: E-mail: martinetz@inb.uni-luebeck.de

Published in Proceedings SPIE, Vol. 8667D, Mobile Computational Photography, 2013.

of poses. As input devices we use the Microsoft Kinect and the PMD CamBoard, the algorithm being able to run on any 3D capable camera. A number of enhancements are made before and after the SOM fitting to correct for artifacts such as elongated fingers, and generally to achieve greater robustness.

We propose a marker-less pose estimation method based on a skeleton-like topology consisting of nodes connected by branches, which approximate the shape of the desired object (in this particular case the user's hand or body). The topology is fitted using a self-organizing map algorithm. The processing and fitting is done in 3D, which has several advantages over 2D silhouette processing. Each step of the algorithm will be discussed in the next sections, as well as a particular implementation of the algorithm on an embedded system and various optimization methods that enable the algorithm to run in real-time.

2. STATE OF THE ART

The most commonly used methods to gather accurate data for skeleton tracking are marker-based motion capture systems in the case of the whole-body skeleton or by the use of a "data glove" for hand pose estimation.[5] These methods are cumbersome, can be used only in controlled environments, and are not suitable for the general consumer market.

Thus, marker-less pose estimation is a heavily researched area in image processing - a technical review published in 2006[6] cites dozens of papers on the topic of hand pose estimation. This is mainly because of the fact that the next generation of electronic devices take into consideration new, intuitive ways in which the user will interact with them, such as controlling appliances only with our motion,[7] playing computer games without a keyboard and a mouse,[8] even figuring out the emotional and physiological state of the users and acting accordingly[9](for instance adjusting the lighting based on the user's mood or reacting when the driver of a car is tired).

There are a number of reasons why hand pose estimation is in particular a hard problem[6] . First, the hand itself is a complex object, the fingers and palm comprising of 19 bones and over 20 degrees of freedom (most commonly modeled using a 27 DOF skeleton[6]). Second, because of this complexity, the hand projection in images often involves self-occlusions, which makes segmenting the different parts of the hand more difficult. Third, the movement of the hand can be very fast, with a speed reaching up to 5 m/s for translation and 300°/s for wrist rotation;[6] the hand movement coupled with the slow camera frame rate can make consecutive frames have very little in common, making the hand difficult to track. Adding to these is the issue of the environment users are usually in, the algorithms having to cope with various backgrounds and lighting conditions.

Regarding performance, most algorithms described in Erol et al[6] achieve less than 30 frames per second (which we regard as being real-time), with only one exception[10] . Other solutions exist, which leverage the computing power of the GPU in order to achieve high framerates[11,12] , and for the most part existing approaches are aimed at high-performance desktop machines.

Commercially, there are a number of solutions for desktop PCs[1,2,3] , which can track the user's body and that work with existing 3D cameras and provide a framework for body tracking and gesture recognition.

Probably the most widely known body tracking solution is the Microsoft Kinect for XBox360. It employs machine learning algorithms to estimate the user's body posture[13] , which is then used as input for interactive games on the console.

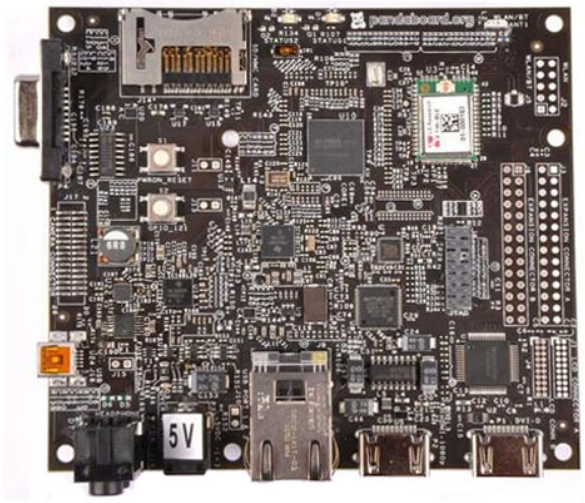
Regarding hand pose estimation, the recently released Intel GestureCam is a near-range time-of-flight camera created in collaboration with SoftKinetic. It allows tracking of the user's hand up to 1 meter. While it does not fully model the hand in 3 dimensions, it does provide the extended fingertips' position and various anatomical landmarks (palm, elbow).

The LeapMotion device[14] promises to allow full 3D tracking of the user's fingers, provided they keep their hands over the device's field of view. The device itself is a small box that needs to be placed on the user's desktop and facing upward. At the time of this writing, the device has not been released.

Finally, there have been some attempts at mobile devices that can track the user's hand or face and respond to simple gestures.[15]

3. HARDWARE

The hardware that is used in this particular implementation consists of two main parts: the ARM development platform and the 3D camera, which provides input data for the algorithm. The resulting images are displayed on a standard computer monitor.



(a) PandaBoard

(b) Microsoft Kinect

(c) PMD CamBoard

Figure 1. (a): The embedded platform used; (b), (c): The two cameras used

3.1 ARM DEVELOPMENT PLATFORM

As a platform suitable for mobile computing we have chosen the PandaBoard (Fig.1a), which is powered by a Texas Instruments OMAP4430 system-on-chip (SoC). The processor itself is used in a number of mobile devices available on the market such as Samsung galaxy S II. The board features a 1GHz dual-core Cortex-A9 ARM CPU, a PowerVR SGX 540 graphics processing unit, which supports OpenGL ES2.0 and enables 3D processing or parallel processing to be done in hardware, 1GB DDR2 SDRAM, two USB 2.0 ports, Ethernet connection and various other peripherals.

The operating system of choice is Linux. We are using a Linaro distribution, which is built and optimized for the ARM architecture implemented in OMAP4430, with the standard toolchain installed.

3.2 3D CAMERA

A necessary step of the algorithm is that of scene segmentation, separating the user from the background. A 3D camera greatly simplifies this process[16] , providing images where objects are separated by depth.

3D cameras have historically been too expensive and of too low quality to provide an accurate representation of the surrounding environment. This changed when Microsoft unveiled in 2010 the Kinect, a peripheral device for the XBOX360 gaming console. The Kinect is able to provide reliable 3D data of up to 15m at a resolution of 640 by 480 pixels, at the same time with (horizontally offset) RGB data, both streams running at 30 frames per second (FPS). A comparable time-of-flight (TOF) 3D camera at that time provided no RGB stream, a depth resolution of 160x120 and a price 20 times higher than Microsoft's device. We have chosen the Kinect as the main camera in this implementation, together with a multi-platform community developed driver, as there are no official drivers for the Linux operating system used on the PandaBoard.

Depth is provided with 11 bits of precision, although beyond 4 meters the accuracy lowers significantly. The minimum distance below which the Kinect stops outputting data is 40cm, at which point the reflected light starts

saturating the infrared camera. Although detail varies with distance and object geometry[17] , we have found that there is enough resolution in the depth map provided by the Kinect between 40 and 100cm so that the fingers are separated, and the hand can be recognized.

The second device used is a time-of-flight camera, the PMD CamBoard. It can output a depth stream at a resolution of 207x204 pixels and 30 FPS. Due to the small form factor of the camera and subsequent size of the LEDs, the distance is limited, accurate readings being made from 20cm up to 150cm. The driver for the camera can provide a "flag" image for each frame, in which invalid or low signal pixels are marked and can be used to filter the depth map so that only valid data will be used.

A comparison between the two cameras is shown in (Fig.2). On the left, the Kinect provides a large but blotchy-looking image. The black areas are shadows, objects that do not reflect infrared well, or areas where the depth computation algorithm failed to produce results. On the right, the PMD CamBoard provides a small and blurred image. At large depths the signal is too weak to be captured back by the sensor, which produces large amounts of noise. There is minimal shadowing due to the LED lights, which are placed closer to the sensor compared to the Kinect. An image artifact of the PMD camera is the thickening of close objects due to the light saturating the sensor. In order to overcome this the integration time can be adjusted so that the sensor captures less light, but as a direct consequence the maximum distance is reduced.

(a) Frame from the Kinect depth stream.

(b) Frame from the PMD CamBoard depth stream

Figure 2. Depth stream comparison: Microsoft Kinect and PMD Camboard.

4. ALGORITHM DESCRIPTION

In contrast with other algorithms, which use only depth maps (usually as a pre-processing step for easier background segmentation)[6] , the SOM approach needs the actual 3D point cloud, which contains the coordinates in space of every pixel in the image, not just the distance from the camera. While this is done automatically by the driver of the PMD CamBoard, the Kinect only provides the depth map. As the intrinsic camera properties (field of view, focal length, pixel size) are known, one can calculate the 3D positions of the points in the 2D depth map.

Because the method requires only the points in 3D space that belong to the subject, they must be separated from the environment background. This is achieved by clipping the points outside a bounding box. For body tracking this works well if the user is alone in the virtual box. For hand tracking we use a different method of background segmentation, taking the closest point from the camera, selecting a rectangular region of 40 by 40 centimeters around it and 20 cm behind it. Regions that are too big or too small are discarded. In practice this works well, as the user's hand is usually held in front of the camera and there are no other objects close by.

In order to estimate the subject's pose, the point cloud must be first reduced to a minimal amount of information. This is done by using a topology representation of the human hand or body. The graph structure

is defined by nodes being connected by edges (seen in Fig.3a for the hand and Fig.4a for the body), which are adjusted in such a way that they will resemble the subject's posture. The models were chosen so they mimic the body (limbs and joints) and hand (phalanges and interphalangeal joints) skeleton for the most important features, whereas the rigid bodies (torso and palm) are modeled as a mesh.

The model fitting phase is done using an algorithm that is usually used to train self-organizing maps. The topology is fitted to the point cloud using an iterative approach with the following rules:[4]

$$\hat{v}^{t+1} = \hat{v}^t + \hat{\varepsilon}^t \cdot (x - \hat{v}^t) \tag{1}$$

$$\tilde{v}^{t+1} = \tilde{v}^t + \tilde{\varepsilon}^t \cdot (x - \tilde{v}^t) \tag{2}$$

Equation 1 shows the rule for the current graph point. First, the Euclidean distances from the current graph node $\hat{v}$ to each of the point cloud samples are calculated, and the closest sample x is chosen. The node is then moved towards the point x by some ε. This is done also for each of the neighbors of the node, $\tilde{v}$ (eq.2), but a different, smaller ε is used ($\tilde{\varepsilon}^t = \hat{\varepsilon}^t/2$).

This method makes the skeleton graph migrate towards the point cloud and stay within its confines. Because this is a "learning" process, the rate with which the points move is adjusted based on the current iteration:

$$\hat{\varepsilon}^t = \varepsilon_i \cdot (\varepsilon_f/\varepsilon_i)^{t/t_{max}} \tag{3}$$

Here, ε_i and ε_f are the initial and the final learning rates, which affect how the SOM responds to changes in the point cloud. Higher initial values provide greater flexibility, but the SOM is prone to unstable behavior like fingertips migrating towards other fingers. After testing the SOM with various parameters, we have set ε_i and ε_f to 0.21 and 0.05 respectively. The indices t, $t+1$ etc. represent the current iteration step.

This approach is robust enough for most poses, but doesn't guarantee that the topology will resemble the actual pose. We provide the example of a subject that brings his arms close to his body as to create a continuous region in 3D space, then extending them to his sides. Sometimes it happens that the extremities of the topology continue to be attracted by the large amount of points in the torso and subsequently remain in that area. This results in the topology following the arm's shape until the last node, which, instead of being in the region of the subject's palm, will be inside his torso, connected by a very long edge.

Generally, the problem fixes itself after some time, but to speed recovery, we introduce an update rule that limits the length of the edges that connect the nodes, effectively limiting cases like the one described. Every update step the distance $d(\hat{v}, \tilde{v}_a)$ is calculated and, if it exceeds a certain threshold θ,

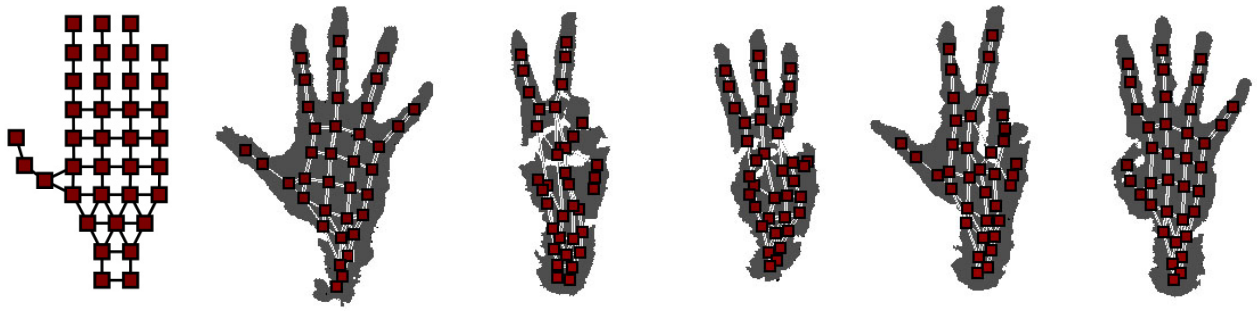
$$\hat{v} = \tilde{v}_a + \theta \cdot \frac{(\hat{v} - \tilde{v}_a)}{\|\hat{v} - \tilde{v}_a\|_2} \tag{4}$$

This rule enforces a maximum distance θ between the current node, $\hat{v}$ and a specific neighbor $\tilde{v}_a$, referred to as an "anchor". This will prevent very large distances to occur between nodes. The anchor is selected for each node as one of the node's neighbors that is closest to the center of the topology.

5. ALGORITHM OPTIMIZATION

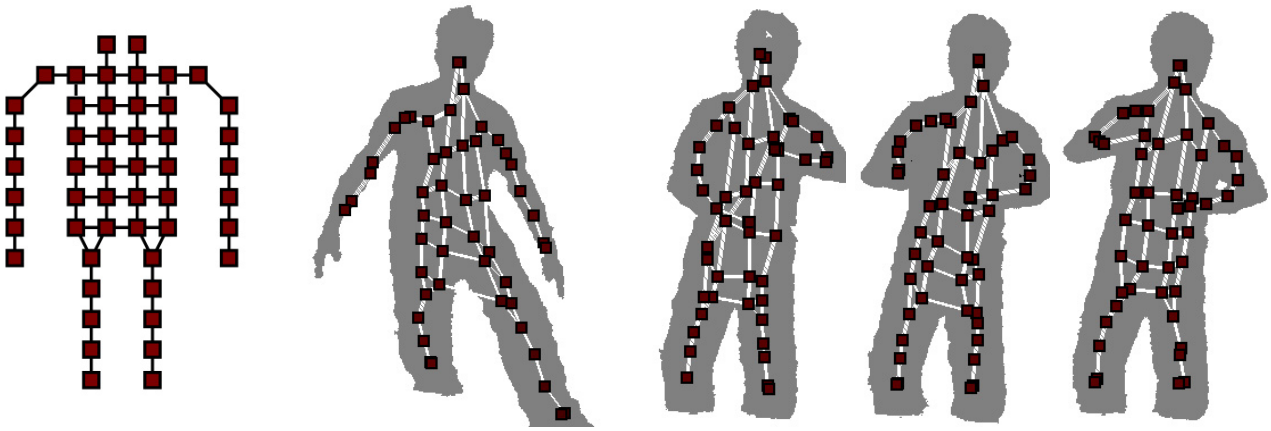
In order to be able to run this algorithm in real-time (30FPS or more) on the embedded system, a number of optimizations have been made. These deal from making sure there are no unnecessary memory accesses to streamlining the algorithm so that it can optimally run on the platform.

The time required to fetch a variable from memory is considerable. We aimed to access memory at little as possible, to prevent the large overhead penalty. In tests, the PandaBoard was able to achieve around 400MB/s for memory copying. This is low compared to the speed achieved when accessing data directly from the cache, at up to 2GB/s. By working only on the data we need, we can keep most of it in the cache, so that the access times are as low as possible.



(a) (b) (c) (d) (e) (f)

Figure 3. (a): The initial (untrained) SOM hand skeleton; (b)–(f): The SOM skeleton for various hand poses.



(a) (b) (c) (d) (e)

Figure 4. (a): The initial (untrained) SOM body skeleton; (b)–(e): The SOM skeleton for various body poses

By converting all the values from the depth image to 3D points we are wasting valuable time, as only a fraction of those meet the requirements to be included in the point cloud. A faster approach would be to filter depth values first, and then, if they pass the criteria, to convert them into 3D points and add them to the point cloud.

The PandaBoard also features a single-instruction-multiple-data (SIMD) pipeline that can significantly speedup computations. Repetitive operations on large data can be significantly accelerated using the NEON engine. By filling up the pipeline, data can be effectively processed in parallel, giving a performance boost to the algorithm.

We can further reduce the time per iteration by a number of small optimizations, such as computing the epsilon iteratively, reducing the computationally expensive exponentiation to a multiplication:

$$\hat{\varepsilon}_t = \varepsilon_{t-1} \cdot \varepsilon_{step} \tag{5}$$

where $\hat{\varepsilon}_0 = \varepsilon_i$ and $\varepsilon_{step} = (\varepsilon_f/\varepsilon_i)^{t/t_{max}}$.

Displaying the results on a monitor often incurs a performance penalty due to the overhead of the libraries used in the process. We solved that issue by not using a window system, instead relying on a console buffer and writing directly to it.

6. RESULTS AND EVALUATION

The end results for the hand and body are shown in Fig.3 and Fig.4, showcasing the method's robustness. It can be seen that the hand tracker is able to cope with missing data (Fig.3c,d as white areas on the palm), the skeleton's topology remaining stable, the fingers being retracted in the palm. This is considered to be correct behavior, as the fingers will be reported as "bent" to a subsequent gesture recognition algorithm.

For the full-body tracker, the topology is robust enough to perform a good fitting over the subject's body, even when there is occlusion occurring. This is shown in Fig.4c-e: it can be seen that when the user crosses his arms in front of him the skeleton retains its geometry afterwards, even if one of the arms occludes the other. This is true also for the rest of the topology nodes, such as the torso. Figure 4b shows how the skeleton tracks the body shape in 3D, following the user's leg even though it's not in the same plane as the body.

Such configurations would have been very hard to track using just a 2D image, particularly because of the juxtaposition of the hands and torso. This issue is resolved by using a 3D camera, which can differentiate between surfaces of various depths. Moreover, note that even though the head is occluded in a couple of frames, the algorithm doesn't lose track of it.

Initially, the performance of the algorithm was mediocre, at around 7 frames per second. The most remarkable speedup was achieved by fully utilizing both cores with separate threads for each step of the algorithm – 17 frames per second, a speedup of almost 150%. By not converting all the data in the frame to 3D the program gained an additional 8 frames per second. By using the NEON SIMD pipeline, an additional 5 frames per second were gained. Other code optimizations like computing the epsilon without the use of exponentiation gained a further 2-3 frames per second. The end result is that the program runs at around 31 frames per second, with room to spare, which was our target.

7. DISCUSSION AND FUTURE WORK

The paper proposes a simple method for human pose estimation from a number of images taken with a time-of-flight camera. This is especially suitable for this kind of camera because they can produce range and intensity data at high frame rates. The self-organizing map approach delivers a very efficient and robust framework, which can be extended to any deformable object that needs to be tracked.

Because of the nature of the SOM, sometimes the tracked skeleton will have erroneous parts, for example a hand being stuck to the other or a skeleton that is upside down. These problems can be solved easily by applying additional constraints in the training phase. Another shortcoming of the method is that it cannot deal with multiple users, because the SOM would attempt to represent a single body shape with the point clouds of the persons sitting in front of the camera. This can be solved by using an improved frame segmentation algorithm, which would detect how many persons are inside the frame and assign a topology for each one, while keeping them separate from one another.

The method can gain further speedups by using the powerful GPU available on the PandaBoard. We have already tested the algorithm on the PandaBoard ES, which uses a next generation Texas Instruments OMAP 4460 and found a performance boost of up to 25%.

We propose two use-cases, which could be tackled by using gesture-enabled technologies, particularly this algorithm, as both require low-power, low-complexity embedded solutions. First, mobile computing faces the challenge of limited interaction capabilities of current and future interfaces. A mobile phone can only get so big before the user becomes encumbered by it. In order to give the user a larger interface area, gesture control can be used. By embedding a small 3D-capable camera inside the phone, a virtual interface can be created and the user can interact with the device via gestures, seeing the results on the screen.

Another possible use for gesture-enabled technology is for wearable interfaces such as head-mounted displays. These have seen a surge of interest in recent times, especially with the backing of Google in the case of project Glass.[18] They aim bring a new level of interaction by presenting data over a head-mounted display, projecting images directly on a viewfinder. In this case gesture control provides a very natural experience to the user, as other types of control interfaces would deny the advantages of using such technology.

A big advantage of this method is the fact that it can work with any kind of 3D data that is supplied by a camera capable of measuring real world distances. From the testing done with the Microsoft Kinect and PMD CamBoard we concluded that the method is robust enough to adapt to any 3D data that is being supplied, as long as it is accurate enough, meaning that the proposed algorithm is able to work with a wide range of cameras, possibly the ones that will be used in the next generation of gesture-enabled mobile computing interfaces.

ACKNOWLEDGMENTS

This work was supported by the Graduate School for Computing in Medicine and Life Sciences funded by Germany's Excellence Initiative [DFG GSC 235/1].

REFERENCES

[1] Omek Interactive, Ltd., "Gesture recognition and body tracking solutions." `http://www.omekinteractive.com/solutions/`. [online; accessed on 03-Jan-2013].

[2] Softkinetic, "iisu sdk." `http://www.softkinetic.com/Solutions/iisuSDK.aspx`. [online; accessed on 03-Jan-2013].

[3] Microsoft Corporation, "Kinect for windows." `http://www.microsoft.com/en-us/kinectforwindows/`. [online; accessed on 03-Jan-2013].

[4] Haker, M., Böhme, M., Martinetz, T., and Barth, E., "Self-organizing maps for pose estimation with a time-of-flight camera," in [*Dynamic 3D Imaging – Workshop in Conjunction with DAGM*], *Lecture Notes in Computer Science* **5742**, 142–153 (2009). http://www.springerlink.com/content/006305183070t383/.

[5] Foxlin, E., [*Handbook of Virtual Environments: Design, Implementation, and Applications*], ch. Motion tracking requirements and technologies, 163–210, Lawrence Erlbaum Associates (2002).

[6] Erol, A., Bebis, G., Nicolescu, M., Boyle, R. D., and Twombly, X., "Vision-based hand pose estimation: A review," *Computer Vision and Image Understanding* **108**(1-2), 52 – 73 (2007). Special Issue on Vision for Human-Computer Interaction.

[7] Bhuiyan, M. and Picking, R., "Gesture-controlled user interfaces, what have we done and what's next?," tech. rep., Centre for Applied Internet Research (CAIR), Wrexham, UK (2009).

[8] Freeman, W., Tanaka, K., Ohta, J., and Kyuma, K., "Computer vision for computer games," in [*Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference on*], 100 –105 (oct 1996).

[9] Kipp, M. and Martin, J.-C., "Gesture and emotion: Can basic gestural form features discriminate emotions?," in [*Affective Computing and Intelligent Interaction and Workshops, 2009. ACII 2009. 3rd International Conference on*], 1 –8 (sept. 2009).

[10] Shimada, N., Kimura, K., and Shirai, Y., "Real-time 3d hand posture estimation based on 2d appearance retrieval using monocular camera," in [*Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, 2001. Proceedings. IEEE ICCV Workshop on*], 23 –30 (2001).

[11] Bayazit, M., Couture-beil, A., and Mori, G., "Real-time motion-based gesture recognition using the gpu," in [*in Proc. of the IAPR conf. on Machine Vision Applications*], 9–12 (2009).

[12] Bierz, T., Ebert, A., and Meyer, J., "Gpu accelerated gesture detection for real time interaction," in [*Visualization of Large and Unstructured Data Sets'07*], 64–75 (2007).

[13] Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A., "Real-Time human pose recognition in parts from single depth images," *Computer Vision and Pattern Recognition* (June 2011).

[14] Leap Motion, Inc, "Leap motion controller." `https://leapmotion.com/product`. [online; accessed on 03-Jan-2013].

[15] Qualcomm Incorporated, "Gesture recognition soccer game demo." `http://www.qualcomm.com/media/videos/gesture-recognition-soccer-game-demo` (aug 2012). [online; accessed on 03-Jan-2013].

[16] Kolb, A., Barth, E., Koch, R., and Larsen, R., "Time-of-Flight Cameras in Computer Graphics," *Computer Graphics Forum* **29**(1), 141–159 (2010).

[17] Andersen, M., Jensen, T., Lisouski, P., Mortensen, A., Hansen, M., Gregersen, T., and Ahrendt, P., "Kinect depth sensor evaluation for computer vision applications – technical report ece-tr-6," tech. rep., Department of Engineering Electrical and Computer Engineering, Aarhus University, Denmark (2012).

[18] Google inc., "Project glass google+ page." `http://plus.google.com/+projectglass`. [online; accessed on 03-Jan-2013].