OpenMoCap: An Open Source Software for Optical Motion Capture

David Lunardi Flam
Thatyene Louise Alves de Souza Ramos
Daniel Pacheco de Queiroz
Arnaldo de Albuquerque Araújo
Universidade Federal de Minas Gerais

João Victor Boechat Gomide
Universidade FUMEC

Abstract

Nowadays motion capture is a valuable technique for virtual character animation in digital movies and games due to the high degree of realism that can be achieved. Unfortunately, most of the systems currently available to perform that task are expensive and proprietary. In this work, an open source application for optical motion capture is developed based on digital image analysis techniques. The steps of initialization, tracking, reconstruction and output are all accomplished by the built OpenMoCap software. The defined architecture is designed for real time motion recording and it is flexible, allowing the addition of new optimized modules for specific parts of the capture pipeline, taking advantage of the existing ones. Experiments with two cameras with infrared LEDs and reflexive markers were carried out and the created methodology was assessed. Although not having the same robustness and precision of the compared commercial solution, this work can do simple animations and it serves as an incentive for research in the area.

Keywords:: Motion Capture, Software Development, Stereo Vision, Digital Image Processing, Computer Graphics

Author's Contact:

{david,pacheco,thatyene,arnaldo}@dcc.ufmg.br
jvbg@hotmail.com

1 Introduction

Analysis and motion capture (MoCap) of human movement have been growing consistently over the last decade. Many researchers from different areas are now working with it because of the wide range of possible applications. Following a classification scheme presented in [Moeslund et al. 2006] it is possible to separate those applications into three groups based on their main objective: surveillance, control and analysis.

Surveillance applications focus on examining motion of a person as a sole object or as a group of objects in case of people agglomeration. Determining the flux of individuals in a mall in order to discover some pattern to optimize shops locations is one example. Another one is a model that describes movement behavior in a prison for security reasons.

Control applications interpret motion from an actor and transform it into a sequence of operations. Movies like King Kong (2005) and Polar Express (2004) use actors movements to operate characters. Other instances are animated characters from 3D computer games, like FX Fighter, Fifa and NBA Live series.

Analysis applications generally dedicate on studying a person as a set of objects. Some examples include improving athletes techniques, studying clinical cases and defining elder people body behavior.

Motion capture is a powerful technique for animating virtual characters and controlling them. It creates smooth movements, giving sensation of real ones. In addition, if used correctly, it can speed up animation, when compared to traditional methods like key-frame animation.

Currently, in Brazil, there are just two organizations that have robust motion capture systems for character animation: Rede Globo and RPM Produtora Digital. Every piece of hardware and software from those systems is imported and expensive, costing tens of thousand of dollars. Usually, game companies in Brazil rent them or buy imported MoCap data.

To our knowledge there are only two national solutions called DVIDEOW [Figueroa et al. 2003] and BraTrack [Pinto et al. 2008]. The first one is widely used for gait analysis, but it is post-processed. In other words, it does not support real time previewing and it can't be used to control operations. Besides that, its source code is closed and apparently it's not actively being developed anymore (homepage of the project [Figueroa 2009] is down at the present time). BraTrack, in its turn, can track objects in real time, but it is not freely available.

In order to acquire expertise and reduce costs, we present the development of an open source optical motion capture software for real time uses in this work. It is a standalone solution, that is, every task needed to acquire MoCap data is implemented by the built application. Further, it is modular and flexible, allowing new modules to be easily integrated and optimized, taking advantage of the existing processing chain. Conclusively, it has a simple graphical interface and it is ready to grow by receiving new contributions.

This paper is organized as follows: related work is introduced in the next section, including some history and commercial and academic approaches for optical motion capture. Our methodology is described in depth in Section 3. The software architecture, chosen programming language and libraries are presented in Section 4. Designed experiments and their results are shown in Section 5. Finally, some conclusions and future work are drawn in Section 6.

2 Related Work

The first process considered to be MoCap was done in 1872. Eadweard Muybridge took several pictures of a horse while galloping with many cameras to settle a bet. The question he answered with that experiment was if a horse would take all his feet of the ground while galloping. Indeed, the horse does take all his feet of the ground, see Figure 1. After that, many other analog processes appeared [Menache 2000], but they were all 2D.

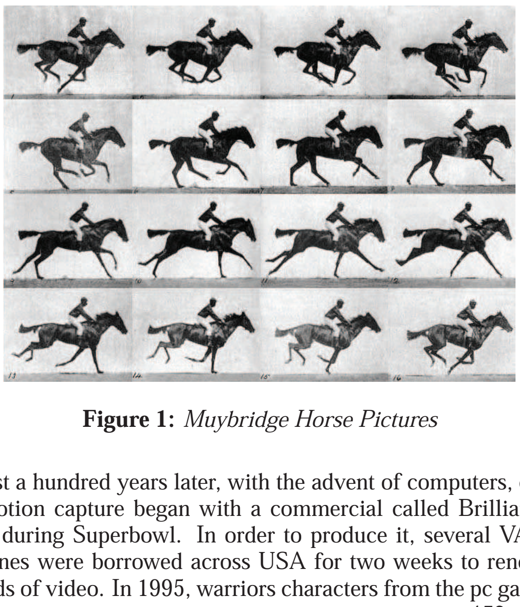


Figure 1: *Muybridge Horse Pictures*

Almost a hundred years later, with the advent of computers, digital 3D motion capture began with a commercial called Brilliance in 1985, during Superbowl. In order to produce it, several VAX 11 machines were borrowed across USA for two weeks to render 30 seconds of video. In 1995, warriors characters from the pc game Fx

Fighter were animated using digital MoCap. Until the present days, several other processes were developed and technologies created to record motion, most of them are detailed in [Kitagawa and Windsor 2008].

Now, we briefly describe three commercial systems that represent the current state-of-the-art in professional optical motion capture. They all cost more than 50 thousand dollars [Inition 2008] and are tightly integrated with proprietary hardware and software. The first one, from Vicon [Vicon Motion Systems 2008], uses passive markers. It is composed by eight T160 cameras, capable of capturing movement in real time at 120Hz with 16 megapixels. Further, it can capture at faster frame rates if camera resolution is reduced.

The second one is called IMPULSE, uses active markers and is made by [PhaseSpace Inc. 2008]. It has almost the same capabilities of Vicon's solution, recording motion at 120Hz with 16 megapixels. The last one is a system that doesn't use markers, called STAGE, made by [Stage 2009]. It is also capable of capturing data at 120Hz, but its estimated precision is equivalent to a 2 megapixels marker system.

While not as robust and precise as commercial systems presented earlier, many papers were published regarding optical motion capture systems. We review and discuss here some of the most recent and relevant.

[Figueroa et al. 2003] report the construction of a brazilian motion capture software for gait analysis, athletes enhancement and other application in the biomedical area. It focuses on tracking markers using a user configurable chain of algorithms based on mathematical morphology, pattern recognition and a Kalman filter. The software also does 3D reconstruction using calibrated cameras and DLT (Direct Linear Transform), but authors don't show experiments for that. Finally, it is post-processed, video sequences must first be recorded for each camera present in the system to obtain results.

[Uchinoumi et al. 2004] perform motion capture without using markers, using just silhouettes of actors iluminated by four cameras, each connected to a computer in a distributed system. Information is processed by four computers that send data to a server through a local network. The server then is responsible for combining every silhouette processed by each client so that 3D reconstruction can be carried out. Authors say that is possible to capture an object at almost 11 frames per second.

[Castro et al. 2006] describe the development of a motion capture system based on passive markers, focused on gait analysis, called SOMCAD3D. It is also post-processed as [Figueroa et al. 2003]. DLT is likewise used for camera calibration and 3D reconstruction. Tracking is done by curve interpolation. In addition, in this work, system precision and accuracy is compared with some other systems specially built for gait analysis.

[Raskar et al. 2007] present a high performance motion capture system with few restrictions regarding the recording environment. It does not use cameras, but instead photo-sensors acting as active markers. They are capable of determining their own position and orientation in space through binary patterns emitted by LEDs placed around the capture volume. Since it has no cameras, expensive hardware is not required and can record movement at rates of 480Hz. Although being composed of reduced cost hardware, their system's assembly and configuration require off-the-shelf electronic and optic components. Finally, it is not possible to capture faces due to the size of the markers.

[Pinto et al. 2008] present the first commercial low-cost marker-based optical tracking system developed in South America. The system is composed by two off-the-shelf USB cameras with custom-made electronic boards with infrared LEDs and reflexive markers. Authors claim that it is possible to capture movement at 60Hz.

Academic systems concentrate on proposing new techniques whereas commercial ones usually advance some already know, mature processing chains as industrial secrets. It does not mean that the reproduction of the papers described here is simple or even possible, principally because some important implementation details are omitted and none of them make their source code easily available.

OpenMoCap is the first step of a bigger project to build a complete and robust optical motion capture system. The main application of the developed solution is the generation of realistic data in real time to animate and control virtual characters. It is the first work that we have knowledge in Brazil of an open source motion capture system created to fulfill that purpose.

3 Methodology

Motion capture can be done using cameras and special selected points. The whole process is complex but can be divided in basically four steps. The first one, *initialization*, regularly is done only in the beginning of the process and relates special given points from a scene with points from a previous defined structure. The second task is called *tracking*, that is, monitor the position of those special points over a period of time. The third one is *reconstruction* or *pose estimation*. Finally, the last one is *output* and consists in outputting data in some special format.

Specifically, when working with 3D models, we must find the correspondence between those special points from each present camera image in the system and apply a triangulation algorithm to obtain their respective 3D coordinates. Obtaining those special points is possible using markers or other local features and heuristics, relative regions positions and skin texture and color. However, the simplest way to retrieve those special defined points is by using markers. Markers are special objects attached to a suit wore by an actor. Also, it is possible to place them directly over the actors body.

Markers can be considered active or passive, depending on their nature. A clear distinction between those classes is that active markers react to external impulses whereas passive markers do not. Furthermore, active markers necessarily have some kind of embedded processing and communication through different kind of sensors and hardware. On the other side, passive markers just have some special properties, like reflecting infrared light. Summarizing, passive markers only help segmenting a region of interest while active ones provide more information about themselves, like their own centroid position (special point) and their identity (semantic relation with the chosen model).

3.1 Initialization

3.1.1 POI Extraction

In order to execute this first step, we must extract some special image points called Points Of Interest (POIs). Each marker in the scene corresponds to a POI in the image. The extraction of those points is accomplished by applying a binary thresholding algorithm. The threshold parameter can be set to values between 0 and 255 (possible intensity values of a pixel in a gray scale image). At first, this procedure is enough to separate POIs from the background.

However, our goal is to separate uniquely detected POIs and also eliminate noise (high intensity parts of the image not related to markers). To perform this, a component connected algorithm with 6-adjacency is applied [Umbaugh 2005]. In practice, the algorithm suffered some modifications to increase its efficiency and reduce noise.

Thresholding is executed in parallel with pixel labelling, so that the image does not have to be looped through twice. As soon as each image element is analyzed, it is marked as being background or object, allowing the connected component algorithm to continue its traditional process. Than, the calculation of the connected components area takes place immediately, enabling the removal of the ones that are probably not markers. This removal procedure is based on user defined limits of minimum and maximum areas. Remaining objects have their centroid calculated. Figure 2 shows the process.

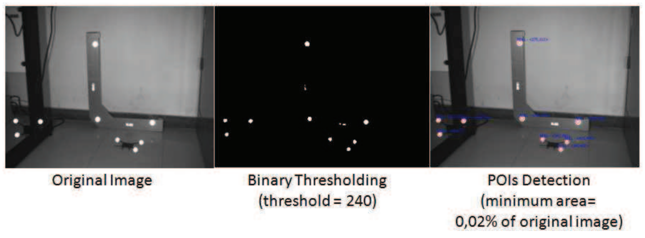

Original Image
Binary Thresholding (threshold = 240)
POIs Detection (minimum area= 0,02% of original image)

Figure 2: *POIs Detection*

3.1.2 Semantic Selection

The next step of initialization is to associate semantics to detected POIs. This is necessary to uniquely define each object with a simple name, like human body parts. OpenMoCap has a skeleton importer compatible with structures written in Biovision Hierarchy (BVH) format [Menache 2000]. An alternative way of semantic input is a simple text file, containing in each line a name. Finally, semantic attribution is carried out manually, in a simple manner, by the graphic interface, shown in Figure 3.

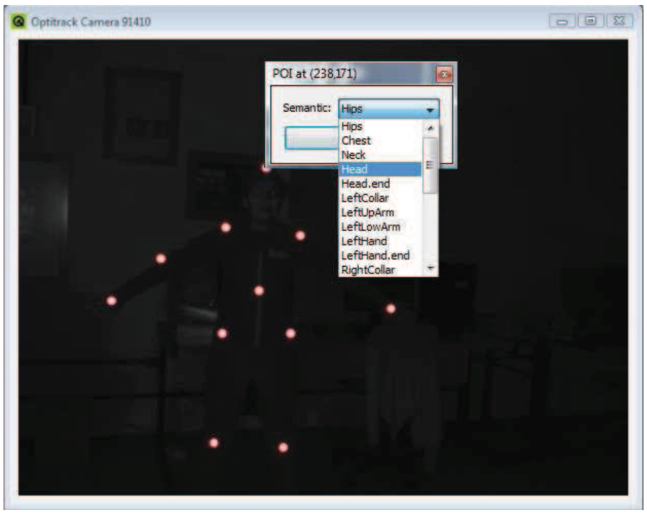


Figure 3: *POIs Semantic Selection*

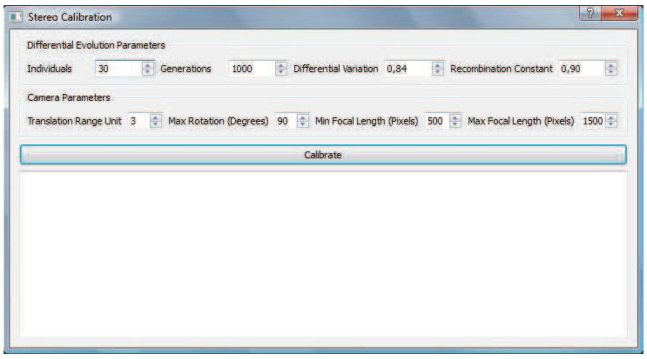
3.1.3 Camera Parameters Estimation

The last step of initialization is the camera parameters estimation. Inspired on [de la Fraga and Vite Silva 2008], our approach to camera parameters estimation is based on modeling this task as an optimization problem, solved by differential evolution. The basic idea is to select some camera parameters as variables and define a cost function over them, using semantic information and POI localization, available from previously described steps. Final result is an estimated projection matrix for each camera, that minimizes that cost function.

Currently, OpenMoCap uses only two identical high quality cameras. Therefore, we ignore lens distortions and chromatic aberrations, simplifying the problem. Hence, the chosen parameters of each camera to be variables of the optimization problem are: focal length f_c, rotation vector $R_c = [r_c^x, r_c^y, r_c^z]$ and the translation vector $T_c = [t_c^x, t_c^y, t_c^z]$.

Variables have their lower and upper limits defined by the user, through the camera parameters estimation interface shown in Figure 4. The first set of parameters controls the differential evolution algorithm, the number of individuals, the number of generations, the differential variation and the recombination constant can be modified.

The second set of parameters effectively defines the lower and upper limits. The Translation Range Unit field defines the maximum translation unit for each coordinate axis, from the first camera to the second. The Max Rotation field determines the maximum rotation


Stereo Calibration

Differential Evolution Parameters

Individuals 30 Generations 1000 Differential Variation 0,84 Recombination Constant 0,90

Camera Parameters

Translation Range Unit 3 Max Rotation (Degrees) 90 Min Focal Length (Pixels) 500 Max Focal Length (Pixels) 1500

Figure 4: *Camera Parameters Estimation Interface*

in degrees (minimum rotation is always 0). Finally, the Min Focal Length and Max Focal Length fields specify the minimum and maximum focal length, respectively, in pixels.

After the definition of the variables and their limits, it is necessary to define the cost function. This function must measure the quality of an estimated solution. Specifically, its inputs are an individual to be evaluated and a set of ordered pairs. Each pair is composed by two POIs with equal semantic and their own centroid coordinates.

Begining with the variables from the target individual of evaluation, two projections matrices are constructed. They represent the estimated orientation and position of the two cameras. Through these matrices and a pair of correspondent POIs, and using triangulation solved by SVD [Hartley and Sturm 1997], we can project a 3D point. Afterwards, this 3D point is reprojected into the image planes. The sum of euclidean distances between those reprojected points and the real centroids of the ordered pair, is called reprojection error.

The total cost of an individual (cost of a possible solution) is obtained by the sum of reprojection errors of all POIs ordered pairs. The smaller the cost the better is the solution. Therefore, our goal is to find the matrices that minimize this global reprojection error.

After this initialization phase, the software is in a valid state, with all data necessary to begin motion capture.

3.2 Tracking

Tracking is the next phase, it is responsible for maintaining POIs' semantics through time. It tries to be as robust as possible regarding observed movement trajectories. This helps the motion capture software to continue in its correct initialization state. Lastly, it is one of the continuous phases of the capture workflow, carried out in every frame.

Tracking starts as soon as a POI receives a semantic, not only when the user requests to begin a motion capture recording section. This is possible because processing is done locally, i.e. only information from the last frame of the same camera is used to define the semantic of extracted POIs in a new frame.

The alpha-beta estimator [Yoo and Kim 2003] is used to appraise POIs' next positions. Figure 5 illustrates how tracking is accomplished. Images with timestamps represent occurred POIs' movements in the scene. While the ones without timestamps represent results obtained by the implemented tracker.

When a new frame is available, the estimated localization of a POI is matched with one of the new extracted centroids. This matching refreshes POI's position with the centroid that has the smaller euclidean distance from the estimated place. Nevertheless, the user can set a maximum allowed distance between these two points, exhibited in Figure 5 by the dotted circles. If no new detected centroid is inside that search region, the new POI's position will be the one predicted by the estimator.

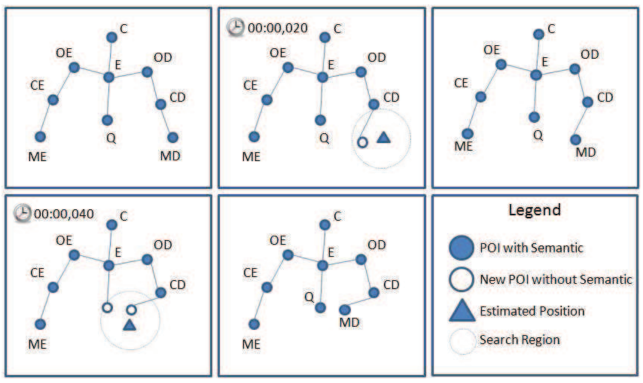


Figure 5: *Sample Tracking Diagram*

3.3 Reconstruction

3D reconstruction is carried out by linear-eigen triangulation [Hartley and Sturm 1997]. This process is responsible for calculating tridimensional POIs coordinates, through their projections on the image planes and their respectives camera matrices. Like the previously phase, this one is continuous, executed frame by frame. But, on the contrary of tracking, it is not local, it is global. In other words, information from all images acquired by the cameras at the same time must be gathered to achieve the desired result.

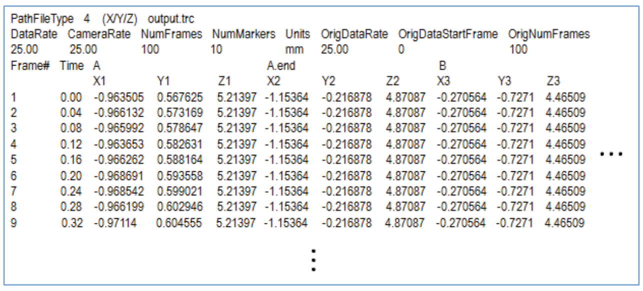
As described earlier in the section of camera parameters estimation, our method does not use tridimensional points with known metric coordinates to estimate camera projection matrices, it is not like a traditional calibration algorithm. Therefore, the final result obtained by triangulation is defined up to a scale factor. Despite this, relative distances between points from the captured structure are maintained. This is often enough to some simple animations and characters control.

3.4 Output

Transforming acquired data by triangulation into usable information is the last necessary phase to complete the motion capture workflow. Basically it runs only once, but it is continuously refreshed after each performed triangulation. In our approach, an output file is generated only after an user requests to stop a motion capture recording section.

There are several types of standard MoCap files [Kitagawa and Windsor 2008]. Fundamentally, they can be classified in two categories: hierarchical and translational. Hierarchical ones contain a defined structure with bones, joints and their relations. Their motion part consists of describing movement as relative rotations, beginning with the skeleton center of mass position.

The other way to represent MoCap data is by describing motion as global translations, without having to define a skeleton. This format is much simpler than the hierarchical one. Since our reconstruction phase produces just point clouds with associated semantic, our software exports data in a translational standard called TRC, developed by [Motion Analysis Corporation 2009]. Figure 6 shows a fragment of a sample file generated by OpenMoCap.



```
PathFileType  4   (X/Y/Z)   output.trc
DataRate   CameraRate   NumFrames   NumMarkers   Units   OrigDataRate   OrigDataStartFrame   OrigNumFrames
25.00      25.00        100         10           mm      25.00          0                    100
Frame#  Time  A                             A.end                          B
              X1         Y1         Z1       X2        Y2         Z2       X3         Y3       Z3
1       0.00  -0.963505  0.567625   5.21397  -1.15364  -0.216878  4.87087  -0.270564  -0.7271  4.46509
2       0.04  -0.966132  0.573169   5.21397  -1.15364  -0.216878  4.87087  -0.270564  -0.7271  4.46509
3       0.08  -0.965992  0.578647   5.21397  -1.15364  -0.216878  4.87087  -0.270564  -0.7271  4.46509
4       0.12  -0.963653  0.582631   5.21397  -1.15364  -0.216878  4.87087  -0.270564  -0.7271  4.46509
5       0.16  -0.966262  0.588164   5.21397  -1.15364  -0.216878  4.87087  -0.270564  -0.7271  4.46509   ...
6       0.20  -0.968691  0.593558   5.21397  -1.15364  -0.216878  4.87087  -0.270564  -0.7271  4.46509
7       0.24  -0.968542  0.599021   5.21397  -1.15364  -0.216878  4.87087  -0.270564  -0.7271  4.46509
8       0.28  -0.966199  0.602946   5.21397  -1.15364  -0.216878  4.87087  -0.270564  -0.7271  4.46509
9       0.32  -0.97114   0.604555   5.21397  -1.15364  -0.216878  4.87087  -0.270564  -0.7271  4.46509
                                              ⋮
```

Figure 6: *TRC Sample File*

The output file is basically divided in two parts, header and data. In the first, we have information about the generated file, the amount of data acquired in a capture recording section per second, the amount of images obtained per second by the cameras, the total number of processed frames, the total number of markers and the used measurement unit. Specifically, this last parameter is set to millimeter, although it has no meaning, since 3D points are recovered without defined scale. The last information in this first part is the title of the captured data columns and POIs' semantics.

The second part of the file is usually composed by many lines of POIs' 3D coordinates, organized according to the header previously described. Each line has the index of the frame which the coordinates were calculated and respective instant of time. Finally, although it is a simple format, it is natively imported (without plugin or scripts) by the 3ds Max software [Autodesk 2009].

4 OpenMoCap

OpenMoCap was built trying to employ most of the good software construction practices described in [Kernighan and Pike 1999] and [McConnell 2004]. Therefore, many architecture and implementation decisions were made to create a high quality, flexible and extensible code.

Figure 7 illustrates the defined architecture by a modules diagram. The separation of modules by threads was done to take advantage of the tendency of modern processors to have more cores. Further, tasks executed in real time such as POI detection, tracking, triangulation and visualization could be made parallel.

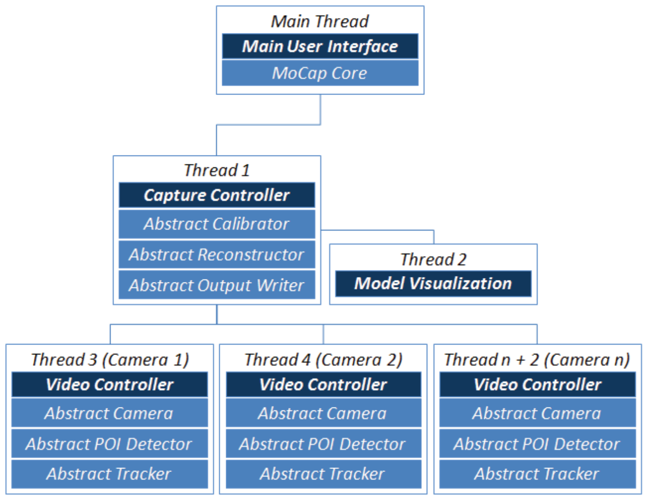


Figure 7: *OpenMoCap Architecture*

The main flow of execution is composed by the application core and the main user interface. MoCap core is responsible for initializing correctly every other flow of execution and their associated modules. Furthermore, it acts as a central information repository keeping track of what is the configuration of the connected cameras, which algorithms are available to perform specific parts of the motion capture workflow and what kind of object will be captured (available POI semantics).

A screenshot of the main user interface is shown in Figure 8. This graphical interface is responsible for showing captured data and for receiving and processing user requests, calling specific functions from the existing controllers. Basically, there are four components that interact with users within the software: the menu, the status bar, camera windows and visualization window.

The menu allows the start and the end of a motion capture recording section. It also informs total capture time and the algorithms being used. The status bar shows the resolution and the frame rate of the cameras being used.

The camera windows display acquired images for each video input device connected to the computer and allow the semantic selection

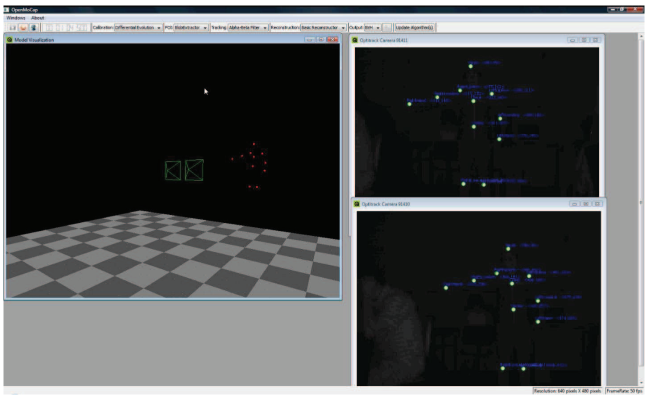


Figure 8: *OpenMoCap Screenshot*

for the detected POIs. The visualization window displays a preview in real time of the motion that is being recorded through a dedicated thread. The Multiple Document Interface (MDI) model was adopted to provide a common and flexible space for all those windows.

Each camera window is processed exclusively by a thread, coordinated by a video controller module. Each Video Controller has an instance of a camera, a POI detection algorithm and a tracking algorithm that are executed sequentially. In Figure 7 this is represented by the modules Abstract Camera, Abstract POI Detector and Abstract Tracker. They define a common and generic interface for the cameras and the 2D processing algorithms cited earlier.

This concept of abstract modules is very important to ensure the software extensibility, one of the goals of this work. If a new camera is to be used and it is not compatible with the current implementation, just some specific methods need to be implemented (like obtaining frames and changing resolution) following the pattern of the abstract camera module to take advantage of the existing processing chain. Another beneficial example is the possibility to substitute the POI detection algorithm with one based on body parts recognition instead of regions intensity. In other words, transform the system into one without markers.

The remaining flow of execution in Figure 7 that has not been described yet is the one composed by the capture controller module. It is the leading contributor for the correct software operation. Effectively, it executes a motion capture recording section, managing and processing data produced by the video controllers, feeding and updating the visualization module. Based on the same principle of abstract modules, the capture controller has an instance of a camera parameters estimation algorithm (Abstract Calibrator), an instance of a reconstruction algorithm (Abstract Reconstructor) and an instance of an output algorithm (Abstract Output Writer).

Until now we have discussed higher level architecture decisions, related chiefly with the concepts of abstraction, generalization, separation of interests and incremental development. In addition to it, some lower level decisions were also made to guarantee profit from existing tools and to obtain high performance.

Given the nature of the built application, an efficient and mature computer vision library was specially usefull for its construction, OpenCV. [Intel Corporation 2009] made it to demonstrate its processors performance. It is written in C++ and it's maintained nowadays by [Willow Garage 2009].

Unfortunately, the existing modules that support cameras in OpenCV are not robust for more than two of those devices and were not developed with object-oriented concepts in mind. Since we want our software to grow and solidly support multiple cameras, videoInput library [Watson 2009] was integrated. Theoretically, it supports up to 20 cameras and is compatible with every video input device that provide a DirectShow interface [Microsoft Corporation 2009].

Regrettably, it was verified that the DirectShow interface provided by the Optitrack FLEX:V100 cameras (used in our experiments) was only able to support one device at a time. For this reason we implemented two new methods in a concrete class for these types of cameras using Optitrack SDK. The software worked flawlessly after this tiny modification, demonstrating in practice the power of the designed architecture.

The graphical interface was entirely built using Qt library [Trolltech 2009]. It is simple, has an open source license and is portable to many operating systems. In addition it provides a easy way to use OpenGL for fast 2D or 3D graphics.

Currently, OpenMoCap only runs in Windows [Microsoft Corporation 2009] because included camera implementations are only supported in this operating system. Besides that, threads were implemented using Win32 API, because Qt threads are slow for our kind of use. Future versions of this software may support other operating systems just by creating concrete classes for cameras and threads, once they were all designed as abstract modules and the rest of the code is portable.

5 Experiments

In this section we present the designed experiments to assess our software and the created motion capture workflow. Furthermore, we discuss their results qualitatively and quantitatively, whenever possible.

In order to obtain quantitative results, we compared our solution with a commercial optical motion capture system made by NaturalPoint [OptiTrack 2008]. Tracking Tools 2.0 can output data in a raw point cloud format and also can display information about tracked 2D POIs, making it a great choice for our analysis. Therefore, we considered its produced data as ground truth.

All experiments were carried out with the same computer, configured with a Intel Core 2 Quad Q6600 processor, 4GB of RAM and a Geforce 8800 GTX. Further, we used the same cameras (OptiTrack FLEX:V100) for both programs. Those devices have a 480p resolution and are capable of obtaining frames at 100Hz. They also have IR LEDs attached and a IR filter that helps POI detection. Finally, identical passive reflexive markers were employed.

5.1 Precision and Stability of 2D Centroids

The main goal of this first experiment is to verify our POI detection algorithm in a static situation. We compare our software results directly with the ones obtained from Tracking Tools, which uses special hardware inside the camera to perform that task. We configured both approaches with the same parameters values shown in Table 1.

Table 1: *POIs Detection Configuration*

Parameter	*Value*
Resolution	640 x 480 *pixels*
Processing Speed	25 frames per second
Intensity Threshold	230
Minimum Area	0,005% of the Image
Maximum Area	0,400% of the Image

A marker was fixed in front of one camera supported by a tripod to make the scene as static as possible. Figure 9 exhibits two graphs showing noise in detected POI coordinates during 50 frames by both approaches.

We applied the normality test of Shapiro-Wilk [Boslaugh and Watters 2008] to a sample of 1000 frames, trying to characterize the generated noise. The result was negative, meaning it is not a gaussian noise. Therefore we analyzed this larger sample using order statistics displayed in Table 2.

The values presented in Table 2 imply that the difference between OpenMoCap and Tracking Tools is very small when detecting POI

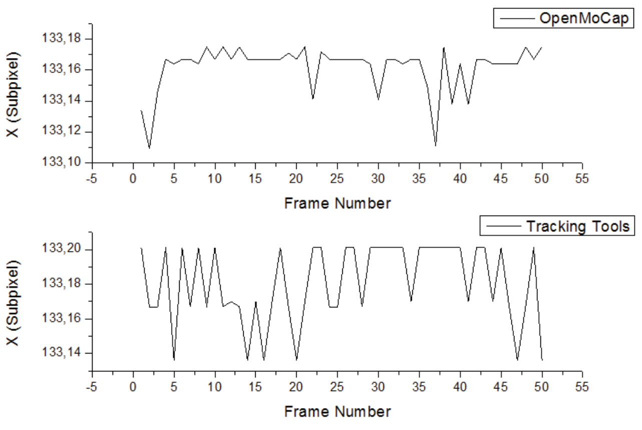


Figure 9: *X Coordinates from a Static POI Obtained by OpenMoCap and Tracking Tools*

Table 2: *Comparative Order Statistics for 2D Centroid Detection*

in subpixels	*OpenMoCap*	*Tracking Tools*
5th Percentile of X	133,164	133,136
Median of X	133,167	133,167
95th Percentile of X	133,201	133,201
5th Percentile of Y	352,826	352,826
Median of Y	352,838	352,844
95th Percentile of Y	352,906	352,856

centroids, from the order of hundredths of pixels. Finally, the calculated percentiles suggest that OpenMoCap has less noise than Tracking Tools in X coordinate, but more noise in Y coordinate.

5.2 Tracking

We evaluated our tracking module by comparing trajectories. The chosen movement for this analysis was from a simple pendulum because its ideal path can easily be described by an arc of a circumference.

OpenMocap and Tracking Tools were configured just like the first experiment, as shown in Table 1. The only difference was the capture speed, in this experiment 50Hz. In addition, alpha and beta values from our tracker were set to 1,0 and 0,8, respectively.

Due to the difficulty of performing the pendulum movement exactly in a plane parallel to the camera's image one, several periods were captured alternating both approaches. Since we consider measures from Tracking Tools our ground truth, we fitted a curve to its results. Figure 10 shows the fitted trajectory and one random period acquired by OpenMoCap.

Two important conclusions can be taken based on this graph. The first one is qualitative, regarding the quality of the motion captured by OpenMoCap. The obtained trajectory is exactly the one expected from a simple pendulum. Slow speed on extremities and faster ones while getting closer to the center. Besides that, the movement is also symmetrical.

The second observation is quantitative and it is related to the existing displacement between detected OpenMoCap centroids and the places that they should appear (points belonging to the fitted curve). Table 3 summarizes this displacement error. Considering that there is friction and the measures could not be obtained simultaneously, those values let us believe that our tracker perfoms like the commercial solution.

5.3 Camera Parameters Estimation

Empirically, it was verified that the implemented algorithm in this work for estimation of camera parameters only works well in situations where the two used cameras are almost in the same plane. In other words, when there is only a small rotation between them.

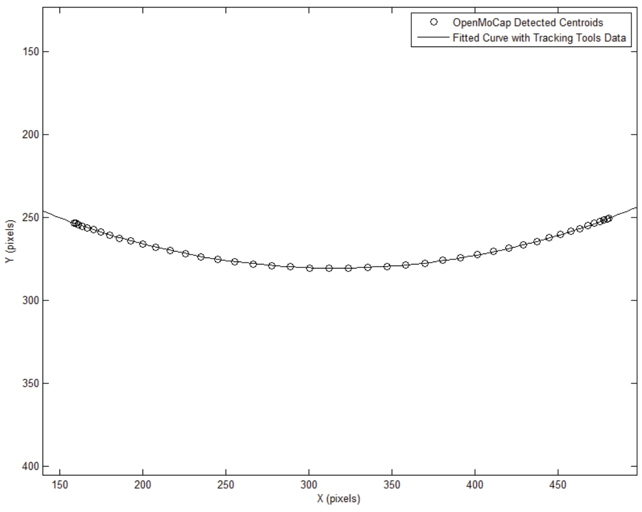


Figure 10: *Tracking Tools Fitted Trajectory and OpenMoCap Sample Trajectory*

Table 3: *Displacement Error*

Error	*Values (pixels)*
Minimum e Maximum	0,004 e 0,254
Median	0,087
85th Percentile	0,125

This happens because the cost function needs many points correspondence to be discriminant. Therefore, when using just a small number of them, the obtained solution is a local minimum. Future work includes recording points correspondence for a few seconds from a scene in order to obtain more projections relations. Finally, the number of estimated camera parameters could be increased, aiming to achieve a better representation of a real camera.

5.4 3D Structure

A direct comparison between 3D coordinates from a structure recovered with OpenMoCap and with Tracking Tools is not possible. The main reason for this is that in our approach we do not have a real scale factor while the commercial approach has.

Despite such difficulty, this work proposes a way to evaluate the quality of the reconstructed structure by comparing distances. This is valid because a point can uniquely be defined in a 3D space through the intersection of four spheres, with trilateration [Doukhnitch et al. 2008].

Figure 11 shows different views of the chosen scene containing a seven marker structure. Two markers on the top of the largest box, two over the black cube and three over a "L" shaped object.

The scene was kept static and was captured by the two programs. The only difference in the starting conditions was that the commercial solution used a third camera (Tracking Tools does not allow less that three cameras for 3D capture) and was calibrated. Our software used just two cameras and the seven points correspondence available in the scene. Figure 12 compares qualitatively the structure obtained by both applications.

Tables 4 and 5 show the calculated euclidean distance between points in the structure recovered by Tracking Tools (in meters) and OpenMoCap.

Considering that both structures are similar and the trilateration principles cited earlier, there must be a multiplicative factor that approximates those distances. Table 6 exhibits the normalized ratios. The lower the dispersion of this multiplicative factor, the better is the quality of the reconstructed structure.

The median of these samples is 0,658. Taking into account only the first and the third quartiles of them, 0,524 and 0,702, respectively,

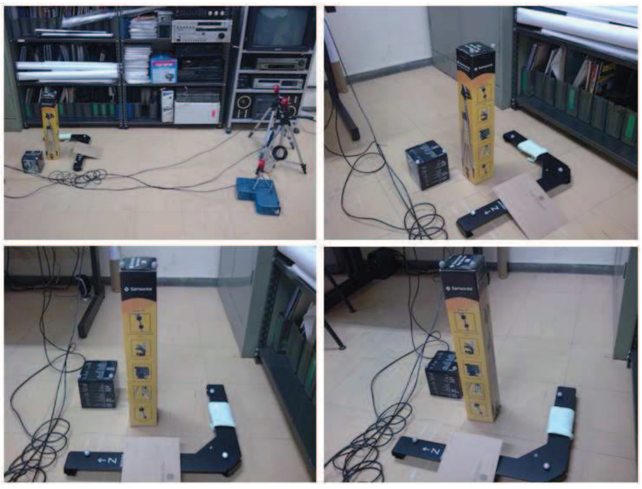


Figure 11: *Scene with 3D Marker Structure*

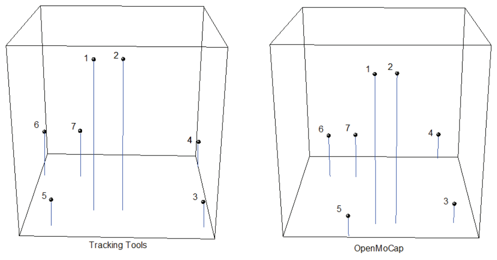


Figure 12: *3D Structure Comparison*

Table 4: *Euclidean Distance Between Points - Tracking Tools*

	1	2	3	4	5	6	7
1		0.075	0.561	0.590	0.498	0.448	0.421
2			0.530	0.557	0.521	0.482	0.440
3				0.313	0.378	0.518	0.435
4					0.510	0.474	0.370
5						0.281	0.283
6							0.106

Table 5: *Euclidean Distance Between Points - OpenMoCap*

	1	2	3	4	5	6	7
1		0,380	2,876	4,162	2,571	3,064	2,883
2			2,708	4,028	2,677	3,192	2,947
3				3,142	1,986	3,297	2,875
4					4,069	2,668	2,176
5						2,685	2,606
6							0,576

Table 6: *Multiplicative Factor Between Distances from OpenMoCap and Tracking Tools*

	1	2	3	4	5	6	7
1		0,504	0,510	0,702	0,514	0,681	0,682
2			0,508	0,719	0,512	0,659	0,668
3				1,000	0,524	0,634	0,658
4					0,794	0,561	0,585
5						0,951	0,916
6							0,540

the maximum difference from the median is 0,134. Therefore the maximum error in this piece of data is around 20%, using Tracking Tools as ground truth. This should be reasonable given our software starting conditions.

5.5 Output and 3D Movement

Even with all limitations of this work, it was possible to capture simple movements from a person with markers in real time at 50Hz. The figure in the first page of this paper shows a key frame of the produced video. The complete video sequence can be downloaded at [OpenMoCap 2009].

5.6 Processing Time

A question that must be answered in this work is if the implemented software architecture is able to record movement in real time. In other words, which the maximum attainable frame rate with the built workflow is. Figure 13 is an area plot that shows the contribution of each step involved in the processing chain to the total processing time for a series of 175 frames. Those frames were specially chosen because of the highest processing peaks observed while recording movement.

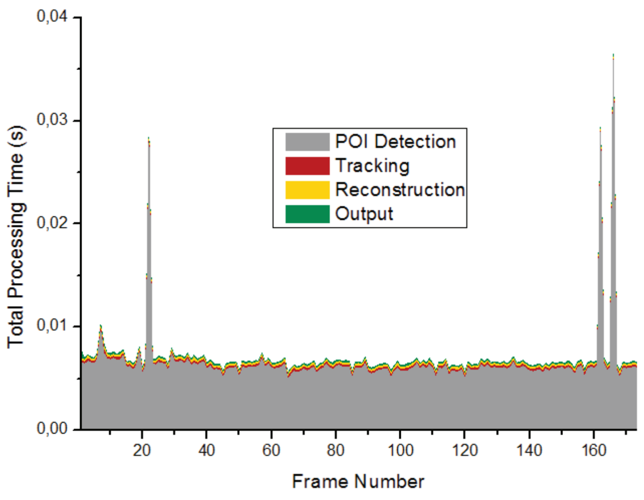


Figure 13: *Area Plot of Total Processing Time in OpenMoCap*

An immediate conclusion of the chart shown in Figure 13 is that almost all processing time is consumed by POI detection. The other steps correspond to minimal portions of the total effort required to record movement. This is why many commercial systems often implement POI detection in hardware.

Another important observation about Figure 13 is regarding processing peaks. The main reason for their occurrence is shared system resources. Processing peaks effectively determine how fast it is possible to record movement, depending on the final use of data. If it is acceptable to lose data from a frame and interpolate trajectories to fix that situation, our software can record movement at 50Hz. On the contrary, if it is not possible to use interpolation, our software is able to record movement at 25Hz.

There is still one contribution of this work that needs to be assessed, the multithreaded architecture. In order to evaluate it, a scene with 20 markers was observed by a variable number of cameras. Figure 14 is a boxplot of the number of cameras used by respective times spent by our software while detecting POIs (most time consuming task performed by our software).

Boxes on the graph represent the intervals between the first and the third quartiles of the samples, i.e. where 50% of obtained measures are located. The larger the number of cameras connected to our software the bigger the boxes are, meaning more data dispersion and less reliability. This is expected and it happens because the operating system shares hardware resources.

Finally, the last interesting conclusion about Figure 14 is related to processing times averages, symbolized by the small squares. Until the fourth camera, only a increase of two milliseconds is perceived on each average. This is much less than the time required to execute POI detection by just one camera, which is approximately six milliseconds. Therefore our multithreaded architecture is validated.

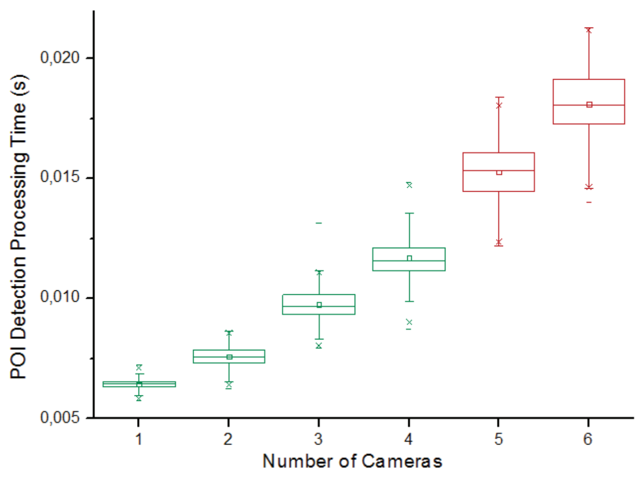


Figure 14: *Boxplot of POI Detection Processing Time by Number of Cameras*

6 Conclusion

In this work, we constructed a real time optical motion capture system from the beginning. The developed software for acquisition and interpretation of data has an open source code. It is a standalone solution, as OpenMoCap includes all the components to process the signal, and its architecture is flexible and extensible as it is possible to modify and add specific modules to improve the workflow. In this way, extensions to allow markless motion capture or to change the camera set can be easily implemented.

This work is part of a project to develop an efficient motion capture system. With this open source system we are making an effort to construct a robust and cheap solution to supply the need for mocap data in character animation in Brazil and in other places where it is not widely used for economical and absence of expertise reasons.

The experimental results show that although it is not a very precise and robust system yet, simple animations can be done with OpenMoCap. Several improvement ideas came up along the development of this software. Unfortunatelly, they haven't been implemented yet due to the lack of time and resources available. As a suggestion to continue this project some of these improvements are listed bellow.

- Define new experiments to better characterize our software.
- Extend OpenMoCap to use more and generic cameras.
- Amend our camera parameters estimation process in order to obtain real scale factor and better precision, allowing facial motion capture.
- Improve our software tracker.
- Perform automatic model initialization.
- Develop POI detection using GPU or distribute processing across a local network with cheap computer nodes.
- Produce output in a hierarchical format, like BVH *Biovision Hierarchy*.

Acknowledgements

This work was supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG).

References

AUTODESK, 2009. Autodesk - 2d and 3d design and engineering software for architecture, manufacturing, and digital entertainment. This is an eletronic document available at: http://www.autodesk.com. Last visited on: May 10, 2009.

BOSLAUGH, S., AND WATTERS, D. P. A. 2008. *Statistics in a nutshell*. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

CASTRO, J., MEDINA-CARNICER, R., AND GALISTEO, A. M. 2006. Design and evaluation of a new three-dimensional motion capture system based on video. *Gait and Posture 24*, 1, 126 – 129. http://dx.doi.org/10.1016/j.gaitpost.2005.08.001.

DE LA FRAGA, L., AND VITE SILVA, I. 2008. Direct 3d metric reconstruction from two views using differential evolution. *IEEE Congress on Evolutionary Computation, 2008 (IEEE World Congress on Computational Intelligence)*. (June), 3266–3273. http://dx.doi.org/10.1109/CEC.2008.4631240.

DOUKHNITCH, E., SALAMAH, M., AND OZEN, E. 2008. An efficient approach for trilateration in 3d positioning. *Computer Communications 31*, 17, 4124–4129. http://dx.doi.org/10.1016/j.comcom.2008.08.019.

FIGUEROA, P. J., LEITE, N. J., AND BARROS, R. M. L. 2003. A flexible software for tracking of markers used in human motion analysis. *Computer Methods and Programs in Biomedicine 72*, 2, 155 – 165. http://dx.doi.org/10.1016/S0169-2607(02)00122-0.

FIGUEROA, P. J., 2009. Dvideow. This is an eletronic document available at: http://www.ic.unicamp.br/ pascual/dvideow.html. Last visited on: July 18, 2009.

HARTLEY, R. I., AND STURM, P. 1997. Triangulation. *Computer Vision and Image Understanding 68*, 2, 146–157. http://dx.doi.org/10.1006/cviu.1997.0547.

INITION, 2008. Motion Capture / Tracking from Inition. This is an eletronic document available at: http://www.inition.co.uk/inition/products.php?CatID_=11. Last visited on: November 26, 2008.

INTEL CORPORATION, 2009. Laptop, notebook, desktop, server and embedded processor technology - intel. This is an eletronic document available at: http://www.intel.com. Last visited on: May 26, 2009.

KERNIGHAN, B. W., AND PIKE, R. 1999. *The practice of programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

KITAGAWA, M., AND WINDSOR, B. 2008. *MoCap for Artists: Workflow and Techniques for Motion Capture*. Focal Press, Burlington, MA, USA.

MCCONNELL, S. 2004. *Code Complete, Second Edition*. Microsoft Press, Redmond, WA, USA.

MENACHE, A. 2000. *Understanding Motion Capture for Computer Animation and Video Games*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

MICROSOFT CORPORATION, 2009. Microsoft corporation. This is an eletronic document available at: http://www.microsoft.com. Last visited on: May 26, 2009.

MOESLUND, T. B., HILTON, A., AND KRÜGER, V. 2006. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding 104*, 2, 90–126. http://dx.doi.org/10.1016/j.cviu.2006.08.002.

MOTION ANALYSIS CORPORATION, 2009. The industry leader for 3d passive optical motion capture. This is an eletronic document available at: http://www.motionanalysis.com/. Last visited on: February 5, 2009.

OPENMOCAP, 2009. Sample capture video. This is an eletronic document available at: http://files.getdropbox.com/u/226543/mocap.avi. Last visited on: July 24, 2009.

OPTITRACK, 2008. Optical motion capture and tracking :: Optitrack. This is an eletronic document available at: http://www.naturalpoint.com/optitrack/. Last visited on: December 2, 2008.

PHASESPACE INC., 2008. PhaseSpace Inc — Optical Motion Capture . Este um documento eletrnico disponvel em: http://www.phasespace.com. Last visited on: November 13, 2008.

PINTO, F., BUAES, A., FRANCIO, D., BINOTTO, A., AND SANTOS, P. 2008. Bratrack: a low-cost marker-based optical stereo tracking system. In *SIGGRAPH '08: ACM SIGGRAPH 2008 posters*, ACM, New York, NY, USA, 1–1.

RASKAR, R., NII, H., DEDECKER, B., HASHIMOTO, Y., SUMMET, J., MOORE, D., ZHAO, Y., WESTHUES, J., DIETZ, P., BARNWELL, J., NAYAR, S., INAMI, M., BEKAERT, P., NOLAND, M., BRANZOI, V., AND BRUNS, E. 2007. Prakash: lighting aware motion capture using photosensing markers and multiplexed illuminators. *ACM Transactions on Graphics 26*, 3, 36. http://doi.acm.org/10.1145/1276377.1276422.

STAGE, 2009. Organic motion: Solutions. This is an eletronic document available at: http://www.organicmotion.com/solutions/stage. Last visited on: February 2, 2009.

TROLLTECH, 2009. Qt Software - Code Less, Create More, Deploy Everywhere. This is an eletronic document available at: http://trolltech.com. Last visited on: January 4, 2009.

UCHINOUMI, M., TAN, J. K., AND ISHIKAWA, S. 2004. A simple-structured real-time motion capture system employing silhouette images. In *Proceedings of the IEEE International Conference on Systems, Man & Cybernetics*, 3094–3098. http://dx.doi.org/10.1109/ICSMC.2004.1400814.

UMBAUGH, S. E. 2005. *Computer Imaging: Digital Image Analysis and Processing*. CRC Press, Boca Raton, FL, USA.

VICON MOTION SYSTEMS, 2008. Motion Capture Systems from Vicon. This is an eletronic document available at: http://www.vicon.com. Last visited on: November 13, 2008.

WATSON, T., 2009. videoInput Library. This is an eletronic document available at: http://muonics.net/school/spring05/videoInput/. Last visited on: January 4, 2009.

WILLOW GARAGE, 2009. OpenCV - Wiki. This is an eletronic document available at: http://pr.willowgarage.com/wiki/OpenCV. Last visited on: January 4, 2009.

YOO, J.-C., AND KIM, Y.-S. 2003. Alpha-beta-tracking index ([alpha]-[beta]-[lambda]) tracking filter. *Signal Processing 83*, 1, 169 – 180. http://dx.doi.org/10.1016/S0165-1684(02)00388-2.