

A Motion Capture System



Daniel Emtén
Johan Karlsson
David Kästel
Mikael Selegård

Table of Contents

Introduction.....	3
Goal	3
Equipment	3
Priorities.....	3
Capture & tracking.....	3
Animation.....	3
Motion Capture.....	4
Background	4
Prerequisites	4
Chosen method.....	5
Constraints.....	5
Method.....	6
Pipeline	6
Acquisition	6
Image processing	6
Error Handling.....	9
Interpolation.....	9
Filtering.....	9
Export to bva-file format	9
Import from bva-file format.....	9
Generate a simple model and animate it.....	9
The viewer application	9
Discussion.....	10
Conclusion.....	10
References.....	10
Appendices.....	10

Introduction

Goal

Throughout this course we've been learning different techniques for dealing with all kinds of modeling and animation matters. Some of them novel and some of them old. This project aims to study one of many methods that animation artists use to generate motion of virtual human beings.

Namely, what could be done with automated procedurals such as motion capture? We wanted to set up a motion capture system that took the data from a filmed sequence and processed it in order to obtain coordinates and rotation that finally could be visualized.

Equipment

This project couldn't involve expensive materials so we had to deal with a standard digital photo-camera to capture video-sequences. This gave us some kind of a "worst case scenario" for our implementation. As tracking devices we wanted nothing advanced, so cheap optical markers were on our minds. To perform the movements we needed an actor, ourselves.

Priorities

As there were two different parts in this project we had to separate our intentions depending on the specific task:

Capture & tracking

- Find an approach to solve the task.
- Trying to extract any kind of tracking point information for a still image.
- Enhance it with moving images.
- Making the captured information into some useful geometrical data such as coordinates.
- Choose, in consultation with the other group, an appropriate format of storing the data.
- Dealing with errors in the data set.
- Automate the acquisition method and tie it together with the animation implementation and build some easy to use GUI.

Animation

- Animate a simple character in Open GL using C#.
- using MC data in a standardized format (bva-file format).

Motion Capture

Background

As humans easily detect and don't accept unnatural motion the animation of "natural objects" must be done as realistic as possible. The most known creatures in our surroundings are ourselves, human beings. Modeling issues including man-like creatures is still developing and, even though recent results show astonishing cloth, hair and skin simulation, we only get fake copies of us.

Fair enough, the most essential parts are known, such as the skeleton. The next step would be to get this thing moving. There are several tools to animate characters. Some of them involve manual (or semi manual) so called keyframing. This gives fine control but the naturalness of motion is not ensured. Moreover the subtle details, that are hard to synthesize with this technique, give the character personality and mood. This leads us to a trade-off between automation and control. Our concern is the automation process.

We want to capture motion of a performer which has sensors attached to the body. The sensor-tracking method could for instance consist of magnetic or optical markers. None of the existing techniques are neither cheap nor perfect. Tracking devices tend to give annoying rates of errors in the collected data. And even if the hardware and software implementations are good, the quality of motion depends on the performer. Real time systems are also rare. Yet motion capture is the leading method in generating sequences of movement for the filming industry and for recording movement databases as part of third party products.



Fig 1. Shows the person who performed all the motion tracked actions of Gollum in the trilogy "The lord of the rings".

Prerequisites

We are not taught in our studies to deal with hardware implementations of media technology problems so this motion capture system could only be solved in programming the interface. That's why we started thinking of finding a way of extracting data from optical markers in a photo. This was known as basic techniques from other courses and seemed pretty reasonable when it comes to programming and material cost.

Several techniques, using different colours, tiny dots, spheres etc. were on our minds. But as two of our group members shortly are to exploring a framework for Augmented Reality applications, AR-ToolKit, in their Master thesis work we decided to find something including that.

Chosen method

We found an interesting implementation of an AR-Toolkit application, done in Matlab, which was based on a couple of advanced image processing steps that computed a 3D representation of a prepared template's position in the image.

The general idea of the system is to take an image that contains the template as input and to get a "Transformation Matrix". This matrix is used to convert the marker coordinates to the camera coordinate system, which is later used by the animation group to arrange the 3D-scene.

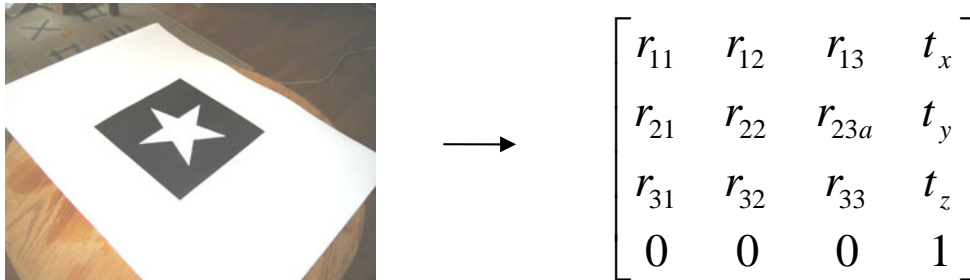


Fig. 2 The photographed template as input which after some processing gave us the "Transformation Matrix"

Constraints

We had to understand and modify the program for our needs. It didn't support more than two markers in the image, which was far too few for modeling a whole body. Even though we just wanted to start with the upper part of the human body, it requested at least half a dozen markers. Furthermore the documentation didn't give us any information about some kind of ideal shape of the templates (markers).

We wanted a stream of images so we had to be able to handle video-files within Matlab and then loop thru the procedure for the marker tracking, which originally only was made for still images. This system would not allow us to occlude any of the markers, so visual contact had to be considered, which gave us quite a limited motion capture system. Either way, it was still interesting enough to study how good it could be with the little equipment required.

Method

Pipeline

Our procedure includes several steps:

- Acquisition
- Image processing
- Error Handling
- Export to bva-file format
- Import from bva-file format
- Generate a simple model
- Animate the model

Acquisition

The modeling team was firstly interested in finding an appropriate data-format, preferably standardized, for storing and accessing the processed geometrical data. After finding and exploring several of these formats we could decide how many markers, joints of the skeleton that we were in need of. (Fig. 3) For speeding up the process and make it possible for the image processing at all to track the markers, the number of them should be as low as possible. Why?

- Less markers means less iterations for the program.
- The more markers the harder for the program to distinguish the uniqueness of a single marker.
- The markers had to be quite big so we could not put too many of them on the actor.

The program works as follows when we build our markers – the making of templates. There is a Matlab-routine that takes an image with a marker as input. The template appearance is printed on almost half the size of an A4 and has to contain connected regions and some simple and easy recognizable figure. The figures and the always squared framing of the template have to be in black and white. The photo with this template is to be taken in good lighting conditions and with the template not orthogonal to the viewing direction. The calculation, described here underneath, gives us the output that will be a binary 2D representation, saved as a proprietary file, normally with exactly the same shape as when it was created in, for instance, Photoshop. A bit awkward step.

We put these markers onto somebody in our group who has to perform some movements. And then, action!

Image processing

For every frame quite a few number of image processing calculations have to be done.

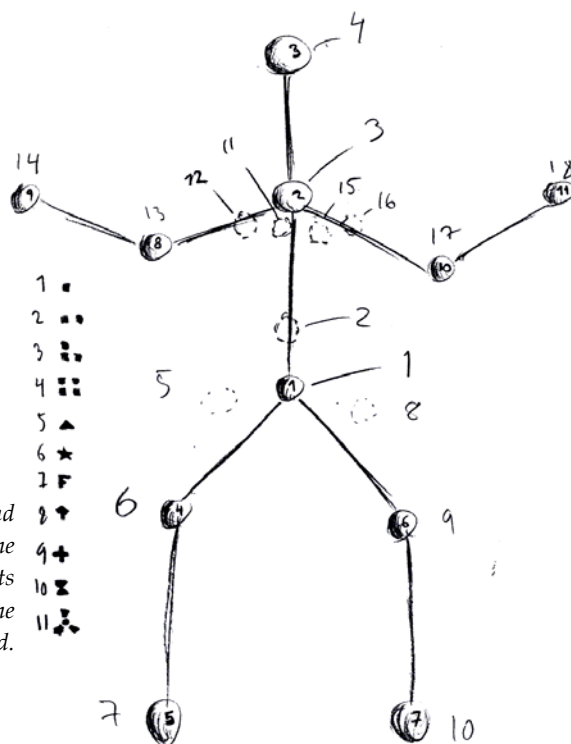


Fig. 3 We had to arrange and document where we put the different markers. These joints were the least necessary for the skeleton-file-format we used.

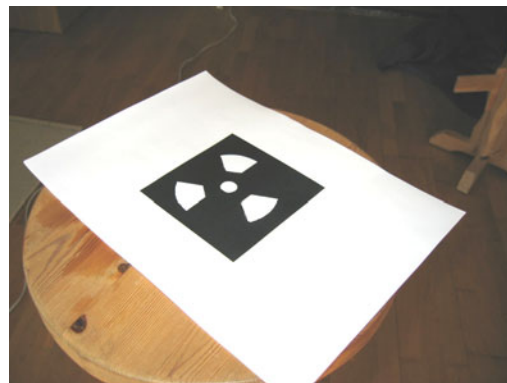
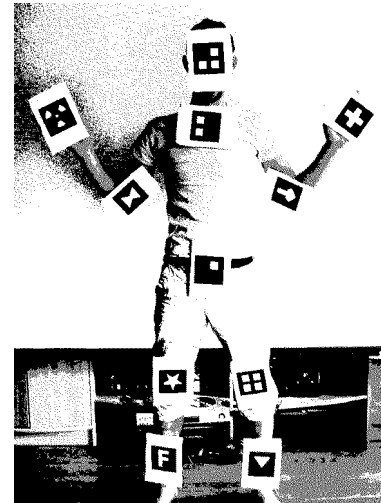


Fig. 4 The templates had to be photographed in a "perspective" mode for the program to be able to extract the template file



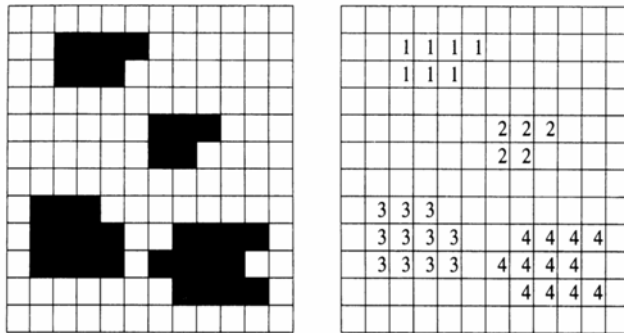
Fig. 5

1. To start with the image is thresholded into a binary version, which will separate the dark parts of the scene from the light ones, thus, our maker from the rest of the scene. The threshold value corresponds to a place in the histogram (pixel value) where the black pixels are separated from white. This value can be changed depending on different lighting aspects.

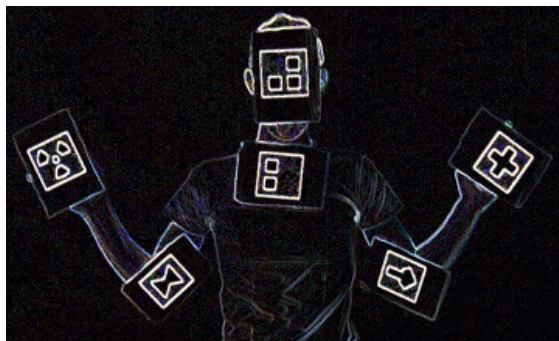


2. Next step decides whether a region is a connected component or not. Works as follows:

- Scan the image to find an unlabelled pixel and assign it a new label, L
- Recursively assign a label L to all its 1 neighbours
- Stop if there are no more unlabelled pixels
- Repeat



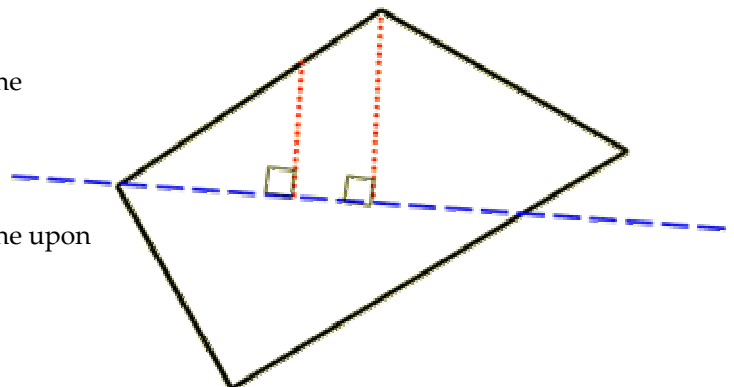
3. Contour seeking: This will represent the image as a chain of pixels.



4. Identification of Corner Pixels

- Fit line through two points on contour
- Find point with maximum distance from this line
- If distance > threshold then we have a corner
- Divide and recurse

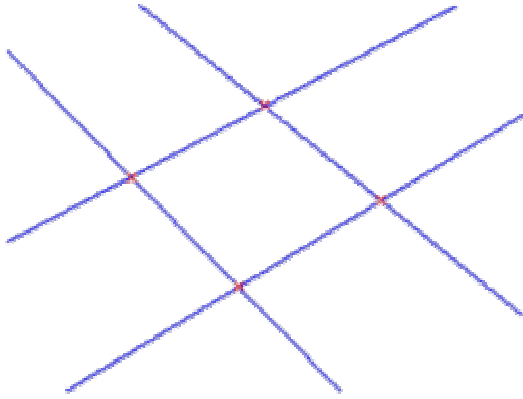
What if the corner pixels are part of false regions? This region has to be rejected; this is done upon



the following criteria. Area, Number of corners and Existence of more likely regions

5. Calculation of the equations of the lines

- Remove points near corners.
- Fit an elongated Gaussian to remaining points using principal components analysis.
- Find corner points by intersecting lines.



6. Intersect Lines to Find Corners => coordinates

7. Identification of template:

- Warp to standard square shape
- Use fundamental theorem of projective geometry
- Calculate correlation



For our intentions we wanted to extract x, y and z coordinates. We could easily extract the x, y for each of the four corners. Since we only wanted one point for the representation, we simply calculated the mean value from the four corners. The z coordinate was a bit more thrilling to acquire. We first tried to extract the z using the transformation matrix used by AR-toolkit. This was a bit tricky and a lot of assumptions had to be made. Unfortunately the values we obtained fluctuated far too much due to sensitivity in calculations. We didn't manage to fix this so we cheated. Instead we calculated how big the Area of the marker was and compared it to a value that we had as $z=0$ in our scene. We also had to scale this value to fit with our x and y values which implies that this will only work when our objects are in the range that we have calibrated for. In our case 3-4 meters from the camera. We also started to calculate the rotation of the markers. For this we used the four corners and created a plane. We could then easily calculate the plane normal and compare it to our view vector. The difference vector could be used to set

the rotations in all axes. Due to our problems acquiring the z-coordinate, the normal also fluctuated too much for usage.

Error Handling

Interpolation

In case a marker was not found we had to consider some error compensation. The easiest way was just to interpolate in the interval between the last two known coordinates. Thus a common linear interpolation, which perhaps has some inconveniences:

- Not realistic
- Most things do not move in straight lines
- Most motions aren't constant over time

But in our case it seemed to work just fine because of the relatively accurate tracking system.

Filtering

As mentioned before we had a lot of problems with the fluctuating Z. We tried to get around this by filtering our signal to avoid abnormal steps. We constructed a simple filter for our needs, but the z-values couldn't be saved. We still use the filter though, to smooth our x and y values a bit to avoid flickering.

Export to bva-file format

We constructed a simple exporter that saved our data to a .bva file as requested from the animation team.

Import from bva-file format

The data from the bva-file was read using standard C# I/O and string manipulation. A data structure for storing initial position, a simple skeleton hierarchy and animation for all the joints was created.

Generate a simple model and animate it

From OpenGL GLU- objects the joints, head, hands and feet were created. Using a predefined definition of the skeleton the joints were then connected. The states of the joints were then animated for each time step by sequentially traversing the data structure.

The viewer application

A simple GUI-application was created in C#. The application can browse and import bva-files and generate an animated simple model. The speed and navigation of the animation can also be controlled by the user.

Discussion

Conclusion

We have done a simple Motion Capture system using cheap and existing technology. The time limit forced us to give up some of our intentions but we are pretty satisfied with the results. This project is nothing to deal with further, but we could of-course use multiple cameras from different angles, to avoid some of the occlusions.

References

<http://mixedreality.nus.edu.sg/software.htm>

Appendices