INF573: Realtime on device motion capture

Matthias Hasler, Guillaume Loranchet

Introduction

Motion capture is becoming an essential tool for film or video game making, especially to create fully virtual worlds. It is often combined with traditional animation but it allows to have a starting point for character animation and be able to capture more realistic motions. Our goal was not to try to implement a professional method, with a 360 system and captors all over the model. Instead, we wanted to explore what was possible to do with a single camera, and low resources; something that everyone could try on their laptop.

The first part we worked on was removing the background to only keep the subject. We found out that most of the methods we later use, do not necessarily need a background removal. We then tracked facial landmarks and shoulders to estimate a 3D position (we wanted our method to work on upper body crops too). The final step was to map the 3D movements to a low poly virtual character (we didn't attempt to replicate any facial expression, only position the head and body parts). Throughout development, a major concern was efficiency. We wanted it to be able to process live videos on a laptop, at a good framerate.

Previous work

A lot of methods use machine learning models to subtract background. For example, the paper: Background Matting [Sengupta et al. 2020] also uses a background image and a video as input, and with a deep learning algorithm, removes the background. Other papers, like this one: A Single Camera Motion Capture System for Human-Computer Interaction by Ryuzo Okada uses a pre-computed body shape that the algorithm will try to match to the video. He first clusters the body shapes in order to find a hierarchy of body poses. Furthermore, he seems to quickly go over the silhouette detection to focus more on the matching. In our work, we try to stay as close as possible to the initial images.

Many CNN based methods exist for pose estimation, like Efficient Object Localization Using Convolutional Networks [Tompson et al. 2006]. However because of the nature of their training sets most of them don't work on upperbody crop, they don't take advantage of temporal consistency, some are expensive to compute, and we could only find a few that mentioned support for 3D poses, with mixed performance.

We don't have at our disposal the resources it takes to train a model, but we can take advantage of some properties of our setup. We can expect the background to be fixed, we can make use of a calibration step, we can leverage recurrences in the video feed to make better predictions

Libraries used

We decided not to use any machine learning algorithms for background removal. The first reason is time and computations. We didn't have time to train a model and we wanted it to be computed in real time. Nonetheless, we still used machine learning libraries, for the facial labelling (with dlib) and the pose estimations (mediapipe, with the PoseNet TensorFlow Lite model - designed to run on mobile devices and the web). However, it was not completely enough to capture the full 3D movement. As we will discuss later, they were not precise enough for depth estimation. Finally, we used meshcat, a WebGL-based 3D visualizer, for the 3D representation. The link between the different platforms were also quite challenging to handle.

Technical implementation

Background subtraction

The main idea we used to remove the majority of the background was a still webcam and therefore a fixed background. Using this, we were able to remove the background fairly accurately by previously taking a picture of the background and took the difference between the current frame and the reference background. We then used a threshold to determine whether a pixel should belong to the foreground or not.

We still had to deal with some flickering pixels. To fix this issue, we used opening and closing morphological operators. Given an image I and a shape S, the morphological opening is equal to $(I \ominus S) \oplus S$ and the morphological closing is equal to $(I \oplus S) \ominus S$. $\ominus$ is the erosion operator and $\oplus$ the dilation operator.

Another technique we could have tried consists in learning the mean and standard deviation for each background pixel from a short clip and process each pixel individually.

The fact that we have a strong prior knowledge of the shape also helps a lot: we are looking for one connected and closed component (we also know that there is a larger part at the shoulders level, smaller for the neck, and a bit larger for the head). Finally, we computed edges to help fill the mask in areas where the background and the foreground had the same color.

Pose estimation

We experimented with scikit-image.skeletonize. Although the framerate didn't drop noticeably, the expensive nature of the algorithm used consumed the majority of the computation time. It was implemented in Cython, but without multithreading, we didn't bother optimising and eventually dropped this component. The resulting topological skeleton was not very stable and appeared hard to analyse in most situations. It delivered promising performance for extended arms, we thought of using a similar approach to estimate the position of the wrists and elbows with more robustness.

We ended up using Google MediaPipe to track the pose. The model was good at tracking the x and y positions but not for the z-coordinates. The 2-step algorithm they used is quite interesting: they locate the head once, this helps finding the other keypoints. Afterwards, they only track locally each keypoint with a Recurrent CNN. We found the computational graph interface of MediaPipe compelling, but we didn't have time to learn the language.

Side note: we assumed that the user would be at a fixed distance from the camera, but we could have added a subroutine that evaluates this to feed it back into the algorithms.

Facial landmarks

First, we had to detect the face. We tried with Haar Cascade, one of the first object detection methods, but it was not robust enough concerning head movements. A Neural Network could have been better but is a bit expensive in terms of computer resources. At the end, we used Dlib, a C++ library that provides machine learning models for detecting facial features and worked well for our needs.

Orientation

We believe that orientation is as important, if not more, than position for our application.

From the 68 facial landmarks the dlib model extracted, we only kept 3 of them: the nose and the two ears. When turning the head, one of the ears might be hidden by the face. Thus creating some problems if the mathematical models computes the 3D geometry based on all the keypoints. We also tried interpolating the ratios nose to ears which gave mixed results. Finally, we hardcoded a simpler method by calculating all 3 axes of rotation from the detected keypoints, as its results were satisfying.

Finding the orientation of the upper body (mainly the shoulders) is quite challenging as the shape could differ depending on the view from the camera (if the model is sitting, standing, the height of the camera etc). From 2D shoulder keypoints, we can accurately represent one axis of rotation, the roll (for example if the left shoulder is higher, we can deduce that the body is leaning toward the right) but not the most interesting one, the yaw (to know if the model is looking toward the left or the right of the camera).

We wanted an ensemble method to make reliable predictions. We need some kind of model to tie orientation to the image we are seeing. We found some correlation between the brightness of some regions and the orientation of the body, but it didn't seem enough for our application. With some clothing, we can accurately track keypoints, and use it to find the chest's orientation.

Smoothing

For our predictions to look realistic, we needed to apply some smoothing. Since many predictions were at the pixel level, we suffered from the relatively low resolution of the video stream, and most predictions were noisy.

On the other hand, we also wanted our model to be responsive. An idea we had was to use dynamic smoothing, estimating movement basic features, for example the difference between consecutive frames or their blurriness. We can then average the previous and new prediction according to this ratio. When standing still the predictions are more stable, when the subject is moving fast, our predictions keep up.

Application

To best showcase our acquisition pipeline, we wanted to create a virtual character mirroring the users' pose. For simplicity sake, we created a boxy character. With meshcat, ZeroMQ sockets are used to communicate each pose in realtime. The character is then displayed using threejs in the browser.

We didn't have pose recognition in mind while doing this project, but the extracted 3D keypoints and the joint structure greatly simplify the subsequent processing.

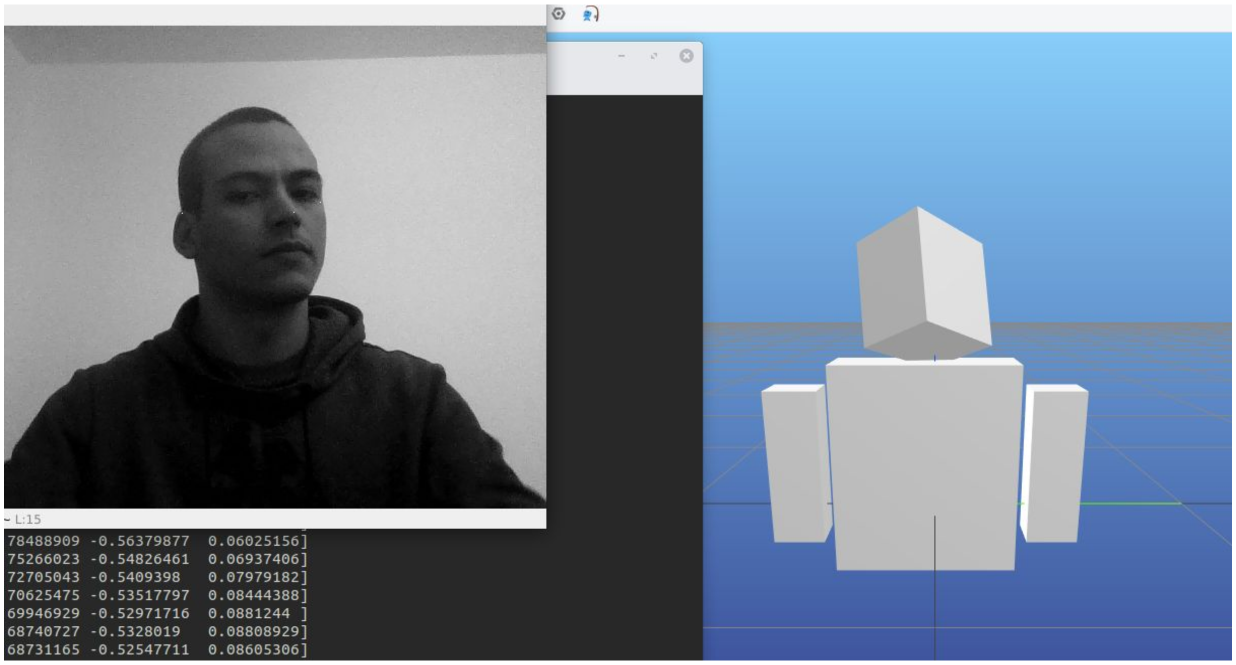
Pipeline / settings

We mainly worked with Jupyter notebooks and Python scripts. We wrote a templated opencv pipeline that enabled us to quickly test modules, and display various outputs. To run the main pipeline, use the file pipeline_1.py.

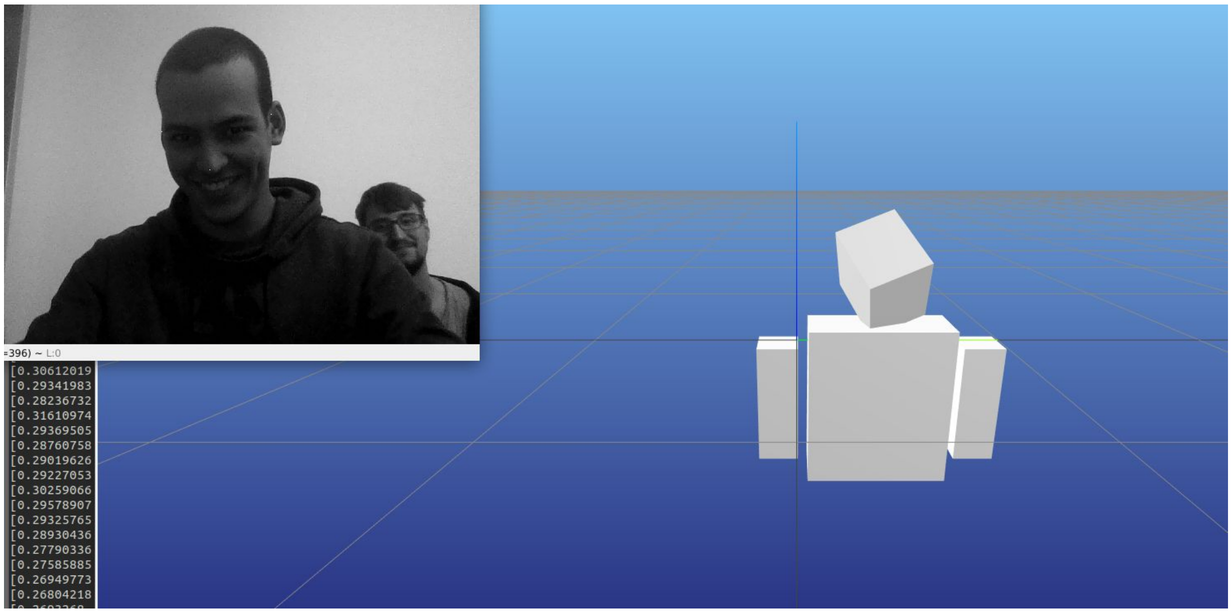
For better results, we fixed the camera's aperture. We were unsure at some point if we should do a real time processing or work on already recorded videos. The main issue was the risk of frame rate decrease for the interactive case. However, we thought that I was more relevant to stay with live processing, as the goal was to be able to interact with virtual characters. It was also useful to have real time feedback to try different positions and immediately see the result. Concerning the resolution, we kept it as default (480p) as we found that it was a good tradeoff between precision and processing speed.

Conclusion

Based on a single camera setup, and with no prior knowledge or captors over the model, we managed to create a real time interaction with a simple virtual character. We were able to make it move over the x and y axis and rotate his head accordingly. We based our predictions on different landmarks and pose estimations for notably the face, the head and the shoulders. We initially wanted to combine various estimators to get the best possible result. Mathematically speaking, it would have been done by multiplying likelihood estimators for the position. However, in practice it could be sufficient to use a discretized prediction space, or combine estimators. We didn’t quite finish what we had in mind, mainly everything related to the arms but our results look promising.



```
~ L:15
78488909 -0.56379877  0.06025156]
75266023 -0.54826461  0.06937406]
72705043 -0.5409398   0.07979182]
70625475 -0.53517797  0.08444388]
69946929 -0.52971716  0.0881244 ]
68740727 -0.5328019   0.08808929]
68731165 -0.52547711  0.08605306]
```



```
=396) ~ L:0
[0.30612019
[0.29341983
[0.28236732
[0.31610974
[0.29369505
[0.28760758
[0.29019626
[0.29227053
[0.30259066
[0.29578907
[0.29325765
[0.28930436
[0.27790336
[0.27585885
[0.26949773
[0.26804218
```