Computer Vision
Report
- Motion Capture -

Andrea Cristiano
Artificial Intelligence Systems
Trento, Italy
andrea.cristiano@studenti.unitn.it
229370

Carlo Serra
Artificial Intelligence Systems
Trento, Italy
carlo.serra@studenti.unitn.it
247189

1 Introduction

In this report, we explore the fundamentals and applications of Motion capture (mocap) technology through a series of practical problems presented in the *Computer Vision* course assignment. We will present our solutions, implementation choices, and assumptions that have enabled us to achieve the proposed results. The assignments consist of three different tasks, each with increasing difficulty. These assignments were designed to provide hands-on experience with mocap systems, starting from an initial situation in which mocap data are given, to post-processing and animation. The aim is to develop a comprehensive understanding of mocap technology, facilitating a deep learning process and enabling students to effectively handle and interact with mocap data. The project source code is available on GitHub[1].

2 Motion Capture

Motion capture[2] technology is a sophisticated method used to record the positions of objects or body parts in space over time. The recording process converts this information into digital data, which can be further analyzed, used to produce animations, or integrated with other applications. It is widely employed in diverse fields such as entertainment, sports, and medical research.

2.1 Optical Marker-Based Motion Capture

Motion capture serves as the foundation for various systems that adapt its core concept to suit their specific needs, utilizing different technologies depending on their intended applications. The data utilized in the initial task of the assignment were generated using an Optical Marker-Based Motion Capture system[3]. This technique, perhaps the most prevalent and effective, employs a set of high-speed cameras to track the movement of objects in three-dimensional space via reflective markers placed on key points of the object.

2.1.1 Task 1

The first task of the assignment fits within this context. The goal was to familiarize ourselves with the standard output files of a motion capture system. We were provided with files in three different formats: CSV, BVH, and C3D.

For the CSV files, we developed a custom solution in Python to read their content. From the first file, we extracted the x, y, and z coordinates for each joint of the skeleton, along with their connections. From the second file, we extracted the coordinates of the markers of a rigid body and of a skeleton. We then connected the markers of the skeleton and plotted the data of each object in 3D using the Open3D library[4], allowing us to observe the positions of the various elements that compose the objects in a 3D space as points clouds. Sample frames are shown in Figures 1 and 2.

For the BVH and C3D files, we decided to use some of the libraries suggested during the course to extract the relevant information, specifically MotionCapture-Laboratory[5] and py-c3d[6]. In both cases, the extracted information is stored within dedicated structures designed to be expandable, allowing for the implementation of additional functionalities that can utilize this data.

2.1.2 Occlusion

Occlusion is a common challenge in Optical Marker-Based Motion Capture systems. It occurs when the markers on the object or cameras are temporarily unable to see the markers, or when the markers' reflections fluctuate due to environmental factors. Occlusion happens when markers on a moving object are hidden by another object or body part. The direct consequence is the presence of gaps in the recorded data, which leads to inaccuracies in motion tracking. If the collected data are plotted, a flickering effect can be observed.

To address this issue, we mainly discussed two possible solutions: improving the number of cameras used for tracking the object or using filtering techniques in post-processing to reconstruct/estimate the position of the missing points. Due to the assignment's structure, we could only implement the second solution.

2.1.3 Task 2

Task 2 involves the implementation of two filters: Kalman filter[7] and Cubic Spline interpolation filter[8]. While the first one is mandatory as per the assignment requirements, the second one is a filter of our choice. We also tried a simple linear regression filter, but we discarded it after seeing its very poor performance.

The Kalman Filter is a widely used estimation algorithm applied in many fields, such as robotics and computer vision. The concept behind it is straightforward and can be summarized in three steps: it predicts the next state of the system, uses a measurement to correct the prediction, and then uses the corrected prediction to make a new prediction. This process is repeated throughout the system's lifetime. One of the main advantages of the Kalman Filter is its ability to handle noisy measurements and uncertainties in the system. Additionally, it can incorporate information such as velocity and acceleration to improve the estimation. We used the Kalman Filter implementation proposed by OpenCV[9], assuming uniformly accelerated motion for simplicity.

Spline interpolation is a form of interpolation where the interpolant is a special type of piecewise polynomial called a spline, chosen such that the overall function is continuous and has continuous first and second derivatives. By ensuring continuity up to the second derivative, splines provide a smooth interpolation. This technique is used to construct new data points within the range of a discrete set of known data points. Cubic spline is the most commonly used spline interpolation method due to its balance between computational simplicity and the smoothness of the resulting curve; it's widely used in numerical analysis, computer graphics, data fitting and other fields. We used the cubic spline implementation proposed by SciPy[10].

In conclusion, we can say that both filters perform very well. In particular, the Kalman filter demonstrates its robustness, especially in the initial situations where the absence of points is significant, managing to create a highly reliable trajectory. On the other hand, the spline shows a slight fragility in the same context. However, once a sufficient amount of data is acquired, both filters provide more than satisfactory results. In computational terms, there are no significant differences between the two filters.

3 Animation

3.1 Motion Recognition & Motion Position Estimation

Although they are two concepts that have a wide range of individual applications, motion recognition and motion position estimation are closely linked to each other. In fact, on one hand, motion recognition allows for the detection of movement and, through the analysis of collected data, the identification of important aspects such as the trajectory of an object, the type of movement, and therefore the action being performed. On the other hand, motion position estimation, as suggested by the name itself, allows for the estimation with a certain degree of precision of the position of the object, or parts of it, subject to motion, thanks to the collected motion data. Pipelines composed of both of these processes are very important and find use in a wide range of applications.

3.1.1 Task 3 - Part I

The goal of the task was to obtain the 3D to 2D projection of joint positions onto the camera plane. The task can be summarized in two main parts.

In the first part, we used Unreal Engine 5 (UE5)[11], a real-time 3D creation platform, to create and record an animated scene, Figure 3, using the LevelSequencer component[12]. We imported the animation and a character into UE5, with which we performed retargeting[13]. Finally, we generated two blueprints[14], one for the actor and one for the camera, which we placed into the scene. We animated the scene and collected the necessary information into two separate JSON files. For the actor, we extracted the position, rotation, and name of each bone for every frame. For the camera, we extracted the position, rotation, field of view, and aspect ratio[15].

To collect the actor's information, we used a specific component provided by UE5 called Animation Pose Snapshot[16]. Since UE5 does not natively support exporting JSON files, we used a dedicated plug-in[17].

3.2 Camera Geometry

A fundamental concept in computer vision is the camera geometry. It describes how a camera captures a 3D scene onto a 2D image plane and it involves understanding the mathematical relationships between the 3D world coordinates and their corresponding 2D image coordinates.

3.2.1 Task 3 - Part II

In the second part, we collected the actor and camera data from the json files to project the 3D points of the actor skeleton on 2d frames from the video recorded in UE5. As explained in a Geeks for Geeks guide [18], we first defined the camera matrix, which represents the intrinsic parameters of the camera: fx, fy, cx, cy; we calculated them using the camera data in the json file. To project and plot the points we used the OpenCV library. Since UE5 coordinate system is left-handed and OpenCV's right handed, we first had to switch the coordinate system from left to right. Then, having the camera amtrix and the rotation and translation vectors of the camera, we used OpenCV's projectPoints()[19] to project the 3D points on the 2D animation frame of our choice.

4 Optional Task

The project also included an optional task that we decided to tackle. The aim of this task was to render the provided BVH file in Blender[20], a well-known 3D visualization environment. We had to interact with Blender using its Python APIs to complete this task. Thanks to the Deep-Motion-Editing (DME) library [21] [22], we were able to interface with Blender quite easily. However, it was necessary to modify the DME library to get the job done. This was likely due to differences in the versions of the Python libraries used

compared to those supported by DME, as well as the fact that the project was developed and executed on Windows. Two examples of the results obtained are shown in Figures 5 and 6.

5 Conclusions

In conclusion, we successfully solved all the tasks presented in the assignment, despite encountering some difficulties along the way. These challenges provided valuable learning experiences, enhancing our understanding of Motion Capture (mocap) technology and its applications. Through this project, we had the opportunity to experiment with tools like Unreal Engine 5 and Blender, gaining hands-on experience. This comprehensive approach facilitated a deep learning process, enabling us to effectively interact with and manage mocap data. Overall, this assignment not only reinforced our theoretical knowledge but also improved our problem-solving skills and technical proficiency in the field of computer vision.

References

[1] *Project Repository*. URL: https://github.com/andy295/Computer-Vision-Project.

[2] *Motion Capture*. URL: https://en.wikipedia.org/wiki/Motion_capture.

[3] *Optical Marker-Based Motion Capture*. URL: https://www.motionanalysis.com/biomechanics/what-is-optical-motion-capture/.

[4] *Open3D Library*. URL: https://www.open3d.org/.

[5] *Motion Capture Laboratory library*. URL: https://github.com/mmlab-cv/MotionCaptureLaboratory/.

[6] *C3D library*. URL: https://github.com/EmbodiedCognition/py-c3d.

[7] Rudolph Emil Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.

[8] *Spline interpolation*. URL: https://en.wikipedia.org/wiki/Spline_interpolation#References.

[9] *OpenCV - Kalman Filter*. URL: https://docs.opencv.org/4.x/dd/d6a/classcv_1_1KalmanFilter.html.

[10] *SciPy - Cubic Spline*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CubicSpline.html.

[11] *Unreal Engine*. URL: https://www.unrealengine.com.

[12] *Sequencer Overview*. URL: https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/Sequencer/Overview/.

[13] *Animation Retargeting*. URL: https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/SkeletalMeshAnimation/AnimationRetargeting/.

[14] *Blueprints Visual Scripting*. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprints-visual-scripting-in-unreal-engine.

[15] *Camera EventGraph*. URL: https://blueprintue.com/blueprint/40ynkxz-/.

[16] *Animation Pose Snapshot*. URL: https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/SkeletalMeshAnimation/PoseSnapshot/.

[17] *Json Blueprint Utilities Plugin*. URL: https://unrealdirective.com/tips/json-blueprint-utilities-plugin.

[18] *OpenCV - project points*. URL: https://www.geeksforgeeks.org/mapping-coordinates-from-3d-to-2d-using-opencv-python/.

[19] *OpenCV - project points*. URL: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html.

[20] *Blender*. URL: https://www.blender.org.

[21] Kfir Aberman et al. "Skeleton-Aware Networks for Deep Motion Retargeting". In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), p. 62.

[22] Kfir Aberman et al. "Unpaired Motion Style Transfer from Video to Animation". In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), p. 64.

Appendix

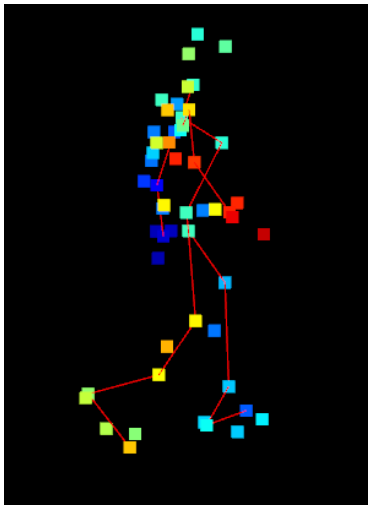

Figure 1: *Task 1: Skeleton plot*

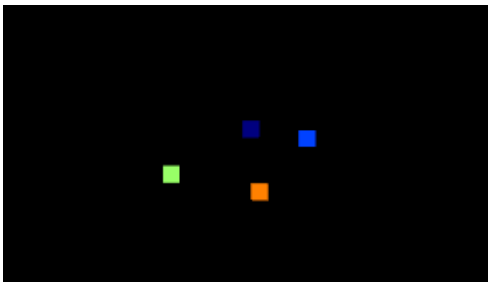

Figure 2: *Task 1: Rigid body markers plot*

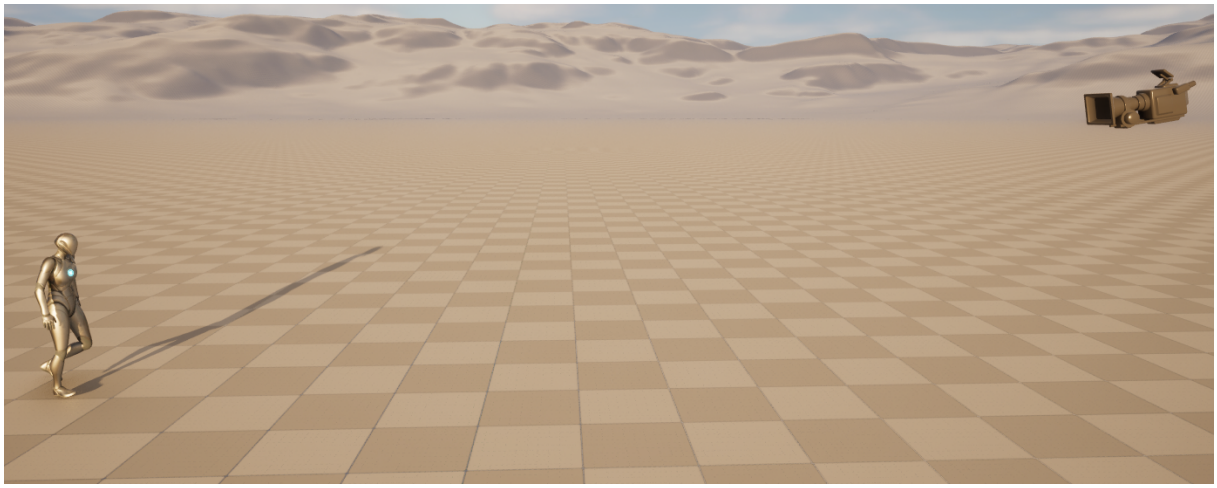

Figure 3: *Task 3: Scene setup*

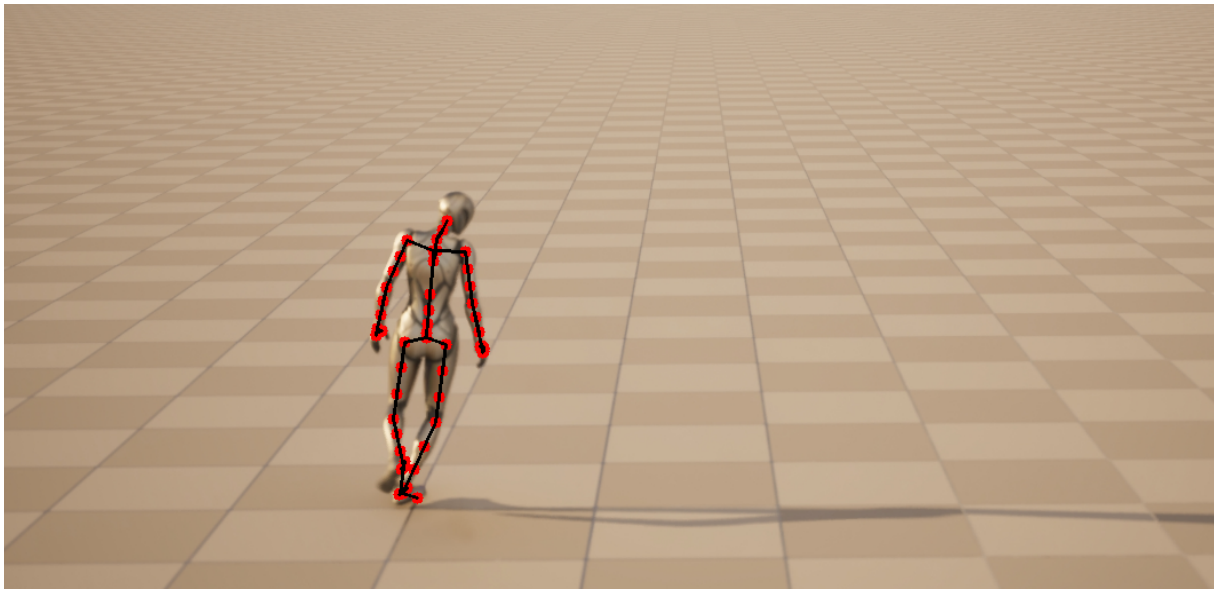

Figure 4: *Task 3: Skeleton projections*

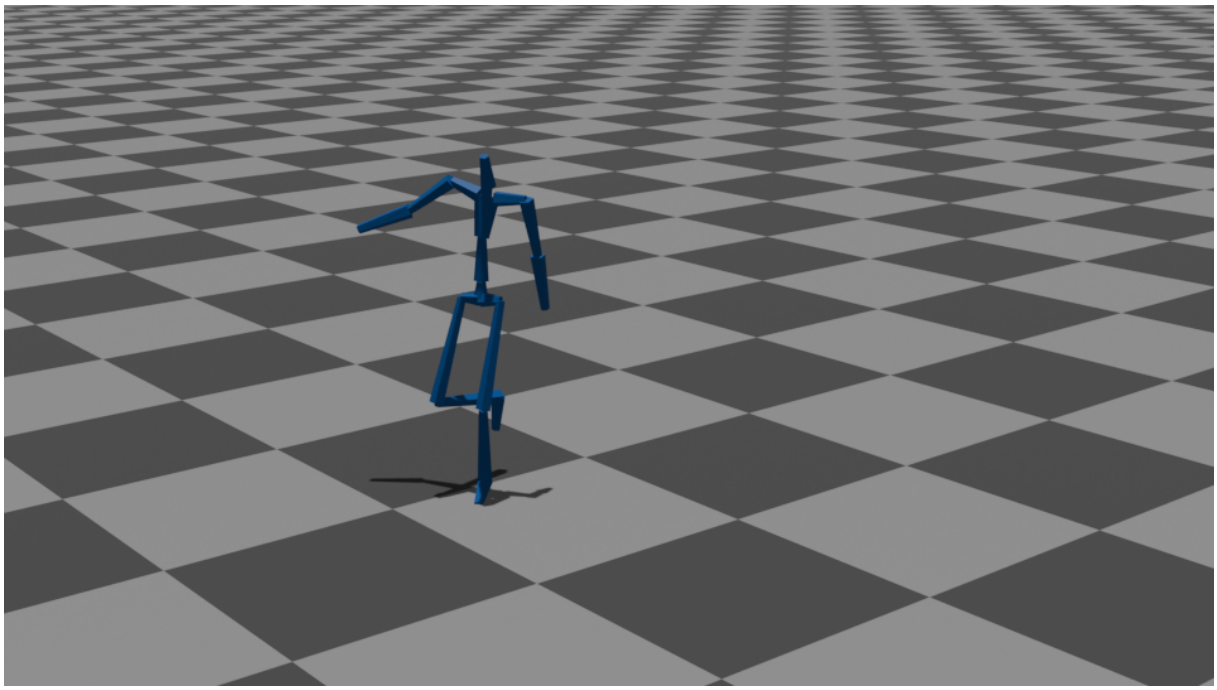

Figure 5: *Optional Task: Blender BVH render*

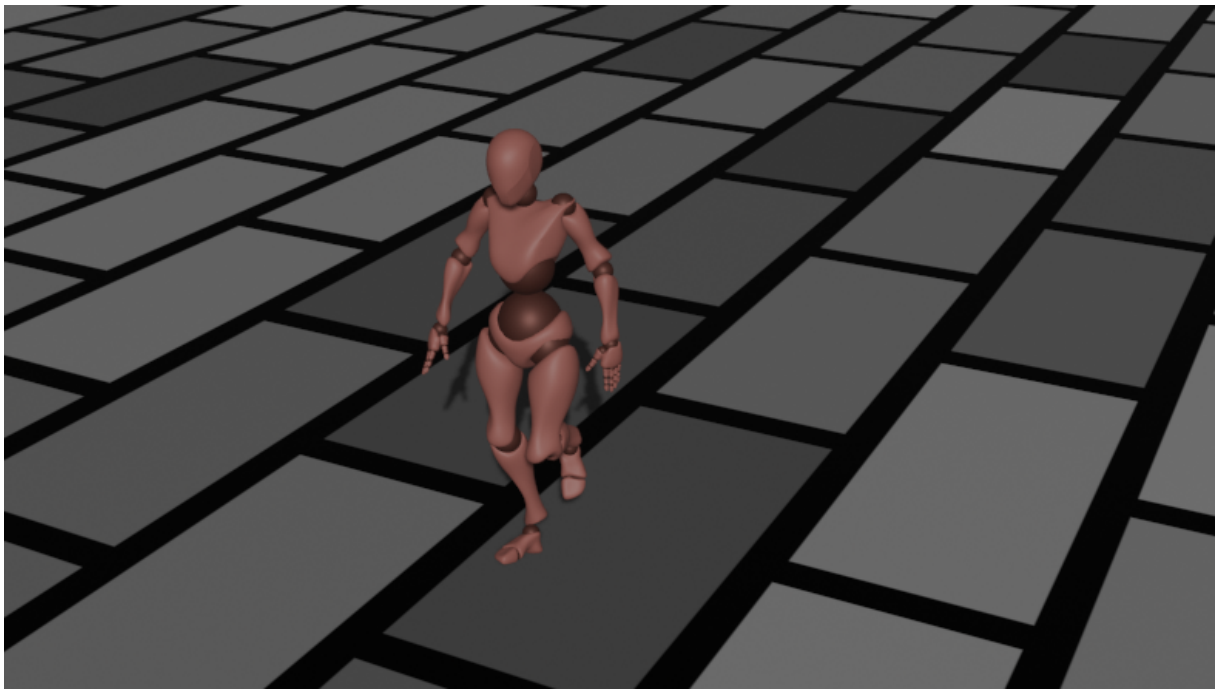

Figure 6: *Optional Task: Blender BVH skinning*