Physics-based model optimization from captured motions

Ilyess Bessaadi

Research Internship report

Master's degree of Computer Science (Image, Development and 3D Technology option)

Supervision: Nicolas Pronost (Univ. Lyon 1, LIRIS), Erwan Guillou (Univ. Lyon 1, LIRIS)

Université Claude Bernard Lyon 1, France
2022

Abstract

Two components are essential for physics-based animation: a character model and a motion simulator. Two different models (*e.g.*, one representing a child and an elderly person) will not be animated in the same way, *i.e.*, their motions will be different (because their physical abilities are different). Our work consists in studying the influence of the model (and more precisely of these physical characteristics) on the motion. We are interested in the human walk of healthy people and people with pathologies. The aim is to be able to compute the modifications to be made to a model according to the difference observed between its healthy motion and its pathological motion.

Résumé

Deux composants sont primordiaux pour l'animation basée-physique : un modèle du personnage et un simulateur de mouvement. Deux modèles différents (par exemple celui représentant un enfant et une personne agée) ne seront pas animés de la même manière, c'est-à-dire que leurs mouvements seront différents (car leurs capacités physiques sont différentes). Notre travail consiste à étudier l'influence du modèle (et plus précisément de ces caractéristiques physiques) sur le mouvement. Nous nous intéressons à la marche humaine de personnes saines et présentant des pathologies. Le but est d'être capable de calculer les modifications à apporter sur un modèle en fonction de la différence constatée entre son mouvement sain et son mouvement pathologique.

1 Introduction

This research internship is financed by the IT department of the University Claude Bernard Lyon 1, and is hosted by the LIRIS (Laboratory of Computer Science in Image and Information Systems). It is a joint research unit of CNRS, INSA Lyon, University Claude Bernard Lyon 1, University Lumière Lyon 2 and Ecole Centrale of Lyon. It has 330 members. LIRIS research concerns a broad spectrum of computer science within its twelve research teams structured into six areas of expertise:

- Data, System and Security (BD, DRIM, SOC and DM2L teams)
- Computer Graphics and Geometry (ORIGAMI team)
- Images, Vision and Learning (IMAGINE team)
- Interactions and cognition (SICAL, SyCoSMA and TWEAK teams)
- Algorithms and Combinatorics (GOAL team)
- Simulation and Life Sciences (SAARA and BEAGLE teams)

During my internship, I joined the SAARA team. The research themes of this team concern the simulation, analysis and animation of complex scenes involving virtual human in motion, with an orientation towards Augmented Reality environments. The targeted applications revolve around the medical and digital entertainment (video games, multi-media, etc.) fields.

It is in this context that my internship subject takes place: "Physics-based model optimization from captured motions". The objectives of the internship are to design and implement a method for calculating a model (or the differences to be made to a model) from captured motions. The computed model should best satisfy the observed motion. By applying this method to a healthy motion and a pathological motion of the same person, we should notice differences between the two models.

The model computation method will be an optimization of a standard character model.

Scientific equipment is available: a Kinect v2 for motion capture, and weights to hang on the body to imitate possible pathological motions. These weights are part of an aging simulation kit, made up of multiple accessories, which together make it possible to understand the variety of everyday gestures of an elderly person. This equipment acts on the three major difficulties linked to aging, namely: the alteration of motor skills, vision and hearing. In our case, we only use equipment that alters motor skills. These are weights that simulate stiffness and muscle weakness caused by aging. The used weights consist of a $9.5kg$ vest, a $2.3kg$ ankle weight and a $1.5kg$ wrist weight.

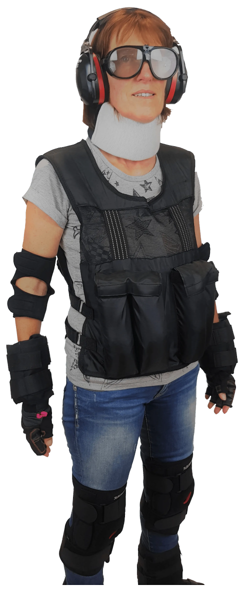


Figure 1: Aging simulation kit

For the progress of this work, we will begin by studying the field of physics-based character animation, in order to have a better knowledge of the existing works, and to orient our work from that. Then we research the existing tools that we will use and get to know them (*i.e.*, Kinect SDK, human locomotion simulator). Next, we plan the organization of our work environment (*i.e.*, all the tools allowing the set up of this work), and implement it. Finally, we will design the optimization of the physics-based model, based on our study of the existing works and our experiments.

2 State of the art

2.1 Human locomotion simulator

For our work, we need a human locomotion simulator. We therefore need to study the different existing simulators in order to understand how they work and to choose the one we will use according to our constraints.

Seen in broad terms, the simulators that we will see work as follows. An articulated physics-based model represents the character to be moved, according to a target motion, then during the simulation, all joints attempt to drive toward their target angles using proportional derivative (PD) controllers.

A controller as described above not has any notion of balance and thus does not produce robust locomotion. That's why every simulator we'll see makes a set of modifications to this design in order to produce robust locomotion.

2.1.1 SIMBICON: Simple Biped Locomotion Control[4]

Input This system receives as input a character's physics-based model and a desired motion. The model should match a biped of human proportion and mass distribution. The data of the desired motion is described by a simple finite state machine or a pose control graph. Each state consists of a body pose representing the target angles.

Use case This state machine can be made up of a few states and be designed manually, but can also be from captured motion. This simulator can therefore correspond to our constraints of use.

Controller In addition to the basic processing of PD Controllers, this control system adds two key components. First, target angles are described with respect to their parent links for all joints, except for the angles of the torso and the swing hip which must be expressed with respect to the world frame. This make the resulting torques be physically realizable without the use of external torques. Also, a feedback term is added to continuously modify the swing hip target angle as a linear function of the center of mass (COM) position and velocity. This provides robust balancing behavior by changing the future point of support.

The employed feedback law to the swing hip is of the form

$$\theta_d = \theta_{d0} + c_d d + c_v v \tag{1}$$

in which θ_d is the target angle used for PD control at any point in time, θ_{d0} is the default fixed target angle as described in the FSM, d is the sagittal distance from the stance ankle to the center of mass (COM), and v is the velocity of the COM.

The feedback gain parameter c_d is therefore important for providing balance during low-speed gaits or in-place (desired zero velocity) stepping.

2.1.2 Generalized Biped Walking Control[8]

Input This system also receives as input a physics-based model that should match a biped of human proportion and mass distribution, but does not really receive any target motion. Since it is produced during the simulation by a motion generator, from:

- A state consisting of joint angle trajectories corresponding to a step.

  This state is used to give the gait style and does not need to be full.

- High level parameters adjustable during the simulation (*e.g.*, step time, step height, coronal and sagital velocity, ...).

Use case This kind of input make manual design of motion easier, but is not compatible with captured motion. This simulator therefore does not correspond to our constraints of use.

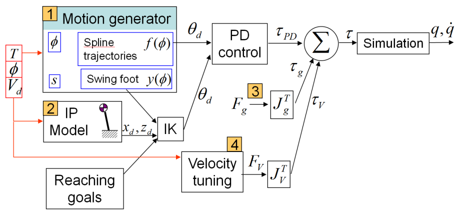


Figure 2: System overview

Controller The control system consists of four key components that are integrated as shown in Figure 2.

Motion Generator The motion generator produces open-loop desired joint angle trajectories. Joint angle trajectories can be modeled relative to their parent-link coordinate frame or relative to the character coordinate frame (CCF). The use of CCF-relative target joint angles introduces additional feedback into the system because of the implicit knowledge at these joints of which way is up.

Comparing to the SimBiCon controller, describing the target joint angles CCF-relative should be a good idea for the torso and swing hip.

PD Control In the same way as for SimBiCon, all joints attempt to drive toward the angles of their angle trajectories for the current instant, using proportional-derivative (PD) controllers.

IP Model A robust balance mechanism is added through the use of foot placement, which is computed with the help of an inverted pendulum model (IPM). The IPM helps achieve motion that is highly robust to disturbances such as pushes.

Using the IPM, the desired stepping point (x_d, z_d) is calculated. Then, target joint angles for the swing hip and knee are computed using Inverse Kinematics, which are then tracked using PD controllers and augmented by gravity compensation torques.

Gravity Compensation The PD-control is augmented with computed-torque gravity compensation. The addition of virtual gravity compensation forces, F_g, and the torques that implement them, $\mathcal{T}_g$, allows for acceptable accuracy tracking to be achieved using low-gain PD-tracking in joint space. The low gains, in turn, help allow for natural, highly compliant motion.

Velocity Tuning Fine-scale control is added by applying a virtual force F_V, on the COM in order to accelerate or decelerate it towards the desired velocity.

2.1.3 DeepMimic: Example-Guided Deep Reinforcement Learning of PhysicsBased Character Skills[16]

Input This system receives as input a character model, a corresponding set of kinematic reference motions, and a task defined by a reward function.

Use case This system is particularly suitable if you have a character who must perform several motions (and not just locomotions). On the other hand, although it is suitable for motion capture, it is not suitable for our constraints of use for several reasons. First, training is required for each new motion set (*i.e.*, in our case each set is made up of a single motion), which will greatly slow down optimization. Also, following this training, a controller is synthesized which makes it possible to best imitate the reference motions, so we will not be able to observe the consequences of the model modifications, since for each of models, a suitable controller is produced.

2.1.4 Flexible Muscle-Based Locomotion for Bipedal Creatures[12]

This simulator is different from others, in that it uses a muscle-based model. The controller therefore does not apply torque to the joints, but muscle excitations instead. This system therefore does not use PD controller unlike the others. On the other hand, the controller using the muscle-based Jacobian transposition approximation to help compute target muscle activations is similar to a joint torque PD controller.

Input This system receives as input a muscle-based model of biped creature, which consists of a hierarchy of rigid bodies, which are actuated using an established dynamic muscle model[5]. It does not take a target motion as input, but the controller produces motion based on the target forward velocity $\tilde{v}_{forward}$ and target heading $\psi_{heading}$ given as input.

Use case This system is adapted to provide locomotions for a multitude of creatures that can be imagined, but does not correspond at all to the use of motion capture, and therefore to our constraints of use.

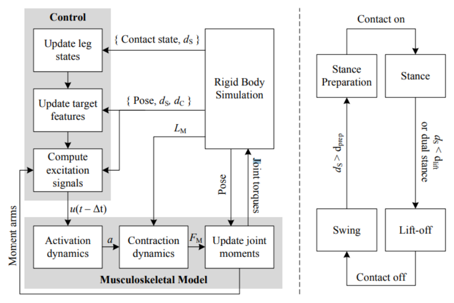


Figure 3: System overview

Controller The goal of the muscle-based control system is to output muscle excitation signals that produce locomotion at a desired speed. To do this, three key processes are applied at each step. An overview of the system is presented in Figure 3.

- First, a finite state machine is updated based on the current leg state.
- Next, a set of target poses is composed for a minimal set of featured body parts. These poses are based on a number of basic feedback rules for speed variation, heading control and balance. All parameters for constructing these poses are found through optimization.
- Finally, the set of excitation signals is computed that make muscle forces drive the featured body parts to their target positions and orientations.

2.2 Optimization strategy

For our work, we optimize the character model, so we have to study the optimization strategies in the field of physics-based character animation.

The muscle model and control model specified in Section 2.1.4, have a large number of parameters, which are set by optimization. The total set of parameters is optimized using Covariance Matrix Adaptation[3], with step size $\sigma = 1$ and population size $\lambda = 20$.

2.3 Termination Conditions

In the case of optimization or reinforcement learning using a simulator, there may be termination conditions. These termination conditions mean that during the evaluation of loss or reward function, simulation is terminated prematurely when failure is detected to save on simulation time and to help prevent local minima.

A condition very suitable for our work is the fall condition. If the character falls, there is no point in continuing the simulation, since it will remain in this invalid state for locomotion. For detect it, we can for example verify if the torso is in contact with the ground[16]. Or, measured and compared the center-of-mass position to its height to the initial state and verify the condition if the measured height falls below a certain threshold[12] (*e.g.*, 0.9).

Here is a list of other termination conditions[12] less suitable to our work.

Heading The target heading $\psi_{heading}$ is compared to the current trunk heading ψ_{trunk}, and terminate if they deviate over 45 degrees. In addition to keeping the character from drifting, this helps avoid a local minimum scenario where a character thrusts its feet forward during a backwards turn.

Self-Collision The simulation is terminated on self-collision, to avoid local minima where a character is unable to take another step because of self-collision.

Leg-Crossing For the same reasons as the self-collision, the simulation is terminated on leg-crossing. It occurs when the coronal left and right foot positions are reversed.

2.4 Loss function

The design of the loss function is an important component of optimization, and therefore of our work. So we have to study the terms of loss or reward functions in the field of physics-based character animation.

2.4.1 Flexible Muscle-Based Locomotion for Bipedal Creatures[12]

The loss function $\bar{E}(\boldsymbol{K})$ to minimize by the optimization, which consists of the following components:

$$\bar{E}(\boldsymbol{K}) = \bar{E}_{speed} + \bar{E}^{ori}_{head} + \bar{E}^{vel}_{head} + \bar{E}_{slide} + \bar{E}_{effort} \tag{2}$$

Each $\bar{E}_m$ is acquired by integrating a time dependent measure $E_m(t)$ over a specific duration t_{max}:

$$\bar{E}_m = W_m * \left\{ \int_0^{t_{max}} E_m(t)\delta t \right\}_{H_m} \tag{3}$$

in which W_m is measure-specific weight, while ${}_{H_m}$ enforces a measure-specific threshold: the value between braces is set to zero if it is lower than H_m. This allows for a prioritized optimization, as heavily weighted terms have greater influence until they reach their threshold. The application of the threshold after integration allows incidental high values to be compensated by below-threshold averages.

Speed

$$E_{speed}(t) = \|1 - \frac{v_{base}(t)}{\tilde{v}_{forward}(t)}\| \tag{4}$$

in which $\tilde{v}_{forward}(t)$ is the target velocity and $v_{base}(t)$ is the forward speed based on the average foot position, updated at each contact initiation.

Head orientation

$$E^{ori}_{head}(t) = \|\boldsymbol{Q}^{-1}_{head}(t)\tilde{\boldsymbol{Q}}_{head}(t)\| \tag{5}$$

in which $\tilde{\boldsymbol{Q}}_{head}$ is the target head orientation, composed of three angles, defined in the transversal, sagittal and coronal plane of the character (applied in that order), and $\boldsymbol{Q}_{head}$ that of the current.

Head velocity

$$E^{vel}_{head}(t) = \|\tilde{\boldsymbol{V}}_{head}(t) - \boldsymbol{V}_{head}(t)\| \tag{6}$$

in which $\tilde{\boldsymbol{V}}_{head}$ is the target head velocity, in the direction of the target heading angle $\psi_{heading}$, and $\boldsymbol{V}_{head}$ that of the current.

Sliding

$$E_{slide}(t) = v_{contact}(t) \tag{7}$$

in which $v_{contact}(t)$ is the average contact velocity.

Effort $E_{effort}(t)$ is the current rate of metabolic expenditure[11].

2.4.2 DeepMimic: Example-Guided Deep Reinforcement Learning of PhysicsBased Character Skills[16]

The imitation objective $r_I(t)$ encourages the character to follow a given target motion $\{\hat{q}(t)\}$, which consists of the following components :

$$r_I(t) = w_p * r_p(t) + w_v * r_v(t) + w_e * r_e(t) + w_c * r_c(t) \tag{8}$$

in which w_m is measure-specific weight.

Joint orientations

$$r_p(t) = exp\Big[-2\Big(\sum_{j \in S_{joints}} \|\hat{q}_j(t) \ominus q_j(t)\|^2\Big)\Big] \tag{9}$$

in which S_{joint} is the set of joints, $q_j(t)$ and $\hat{q}_j(t)$ represent the orientations of the joint j from the simulated character and reference motion respectively, and $\ominus$ is the quaternion difference.

Joint angular velocities

$$r_v(t) = exp\left[-0.1\left(\sum_{j \in S_{joints}} \|\hat{\dot{q}}_j(t) - \dot{q}_j(t)\|^2\right)\right] \tag{10}$$

in which S_{joint} is the set of joints, $\dot{q}_j(t)$ represent the angular velocity of the joint j from the simulated character and the target velocity $\hat{\dot{q}}_j(t)$ is computed from the data via finite difference.

End-effectors positioning

$$r_e(t) = exp\left[-40\left(\sum_{e \in S_{end}} \|\hat{p}_e(t) - p_e(t)\|^2\right)\right] \tag{11}$$

in which $S_{end} = [leftfoot, rightfoot, lefthand, righthand]$, and $p_e(t)$ and $\hat{p}_e(t)$ represent the local position of the end-effector e from the simulated character and reference motion respectively.

Character positioning

$$r_e(t) = exp\left[-10\left(\|\hat{p}_c(t) - p_c(t)\|^2\right)\right] \tag{12}$$

in which $p_c(t)$ and $\hat{p}_c(t)$ represent the position of the character's COM from the simulated character and reference motion respectively.

3 Contribution

3.1 Overview

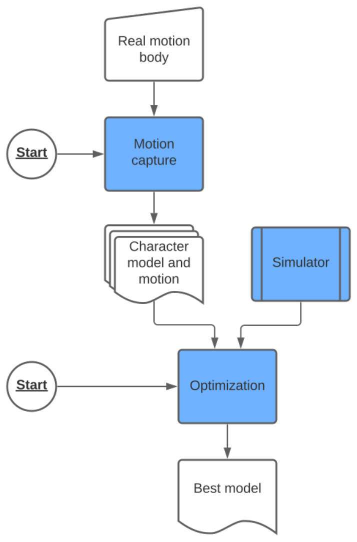


Figure 4: System overview

Our work consists of three key components that are integrated as shown in Figure 4. In a general sense, these components work as follows. A motion capture component produces the character model and motion from the real motion of a person. This data is used by the simulator during optimization. The optimization component optimizes the character model, for this, it uses the simulator for the evaluation of its loss function, and provides the character model that best corresponds to its motion.

There is also a fourth motion visualization component. It takes as input a model and motion in the format of those produced by the motion capture component or for motion, from the simulator component. It generates a character which is a joint hierarchy from the model, then simply translates the character and applies the rotations of each joint from the motion.

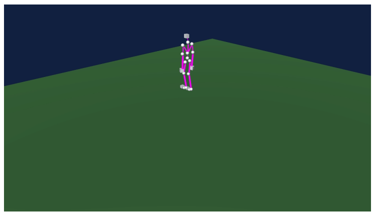


Figure 5: Motion visualization

3.2 Project requirements

The simulation component obviously depends on the chosen simulator, but actually, the whole project also depends on it. Since it is according to what the simulator needs that we will create the model and the motion, and therefore also, that we will know which parameter of the model will have to be optimized.

The way in which the physics-based model of the character is described is at the center of the operating of the simulators. But for our project, no matter how the model is described, we only need to see its consequences on the simulated motion. So we want a simple physics model for the character, in order to reduce the number of parameters to be optimized.

For the simulator, we want one that is already implemented, in order to save a lot of time on setting up the working environment. It only needs to be able to simulate human locomotion, so a very generalized motion simulator is not needed. It must simulate a motion according to a target motion, in real-time or in near real-time (*e.g.*, this is not the case of the simulators mentioned using Reinforcement Learning, which require training for each motion to be simulated) since an optimization can try thousands of different motions, and we don't want to have an unreasonable optimization time.

We will therefore use an implementation of SimBiCon[4] in the "Cartwheel" project. This project accompanies the article "Generalized Biped Walking Control"[8], and therefore also implements its high level controller. But we won't use it since it doesn't really take target motion as input, as explained in the Section 2. We also need to be careful that Proportional Derivative (PD) Controller gains match the strength of a standard human (*e.g.*, too low gains will not allow the character walking, and too large gains will be less constrained by the characteristics of the model).

The physics-based model is therefore a hierarchy of articulated rigid bodies (linked by joints) with different degrees of freedom (DOF), and possessing a mass, a moment of inertia (MOI) and a collision primitive.

3.3 Motion capture

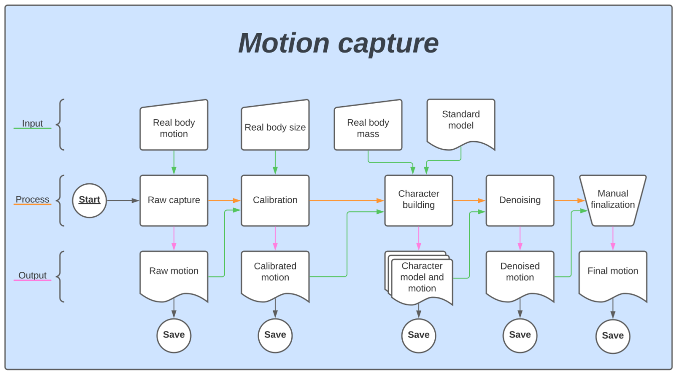


Figure 6: Overview of motion capture component

The motion capture component consists of several processes that are integrated as shown in Figure 6. In a general sense, these processes work as follows. A raw motion capture is therefore first performed from the Kinect v2 (which is made available as mentioned earlier). Then, the capture frame is calibrated in order to respect our constraints of use (in particular of the simulator). Then from this captured motion in the correct frame, we create the model of a character corresponding to the captured person, and convert the raw data of the motion into a motion for the character. Finally, the character's motion is denoise on the joint rotations. A manual finalization is carried out after all these automatic processing in order to save time in the implementation of the character's motion creation. We now go into further details about each of the processes.

3.3.1 Kinect v2 technology

The Kinect v2 sensor is equipped with a color camera and a depth sensor that is capable of measuring the depth of each pixel in its view. The depth of each pixel is computed based on the phase shift of the emitted modulated light and the corresponding reflected light[14].

The Kinect SDK's ability to receive skeleton frames in addition to color and depth frames is one of its most important innovations. The skeleton tracking technology used in Kinect was based on the research carried out by Microsoft Research UK[10].

3.3.2 Raw capture

The Kinect gives us the updated position in real time of 25 joints of the detected bodies. To detect a body, the person must be facing the Kinect or in 3/4, bodies are not detected in profile or with their back to the Kinect. The positions are given in the Kinect's frame, where the origin is the Kinect itself, and where the axes are always organized as follows: the X-axis represents the width, going from the left of the Kinect to the right, the Y-axis represents the height, going from the bottom of the Kinect to the top, and the Z-axis represents the depth, going from the front of the Kinect to the back.

On the other hand, the frame is not always at the same scale. Although the orders of magnitude remain the same, keeping the same position for the Kinect and the person between several captures, the same person may not have the same size in the Kinect's frame. Also, depth accuracy becomes more constant after the Kinect has been operating for a while[13]. The distance varies from $5mm$ up to 30 minutes and becomes then almost constant (more or less $1mm$)[13].

Although we don't need great precision when capturing motion, we decided to pre-heat the Kinect for 30 minutes before using it. Then, we use the Kinect v2 SDK to do our motion capture: we record the raw position in the Kinect's frame of each given joint. The interval duration for which a new motion state is captured can be given by the user, and is initially set to $\frac{1}{24}$.

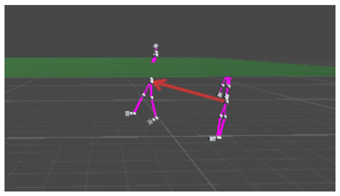
3.3.3 Calibration

Next, we want to describe our motion in a frame that respects the constraints of the simulator. In this frame reference, the X-axis must represent the width, going from the right of the character to the left, the Y-axis must represent the height, going from the bottom of the character to the top and the Z-axis must represent the depth, going from the back of the character to the front. We don't want to describe the motion in the character's frame, but in a world's frame where the character is oriented in the way described during his walk. We therefore calibrate the Kinect's frame.

Since we have to compare several motions, we want the frame to always be at the same scale (*e.g.*, so that 1 unit of each axis corresponds to 1m). We are particularly interested in whether the models of the same person always make the same size. We therefore rescale the frame by the ratio between the size of the real person (which must be given) and that of the captured character.

During simulations, we need to add a collision plane representing the ground, so that the character can walk. We consider that we systematically add a plane having for origin the origin of the frame and for normal the Y-axis. We therefore want to position our character above this ground (*i.e.*, having no articulated rigid body with negative position in Y). Also, to make it easier to compare motions, we want all motions to start at the same position. For example, having the origin of the frame which is between the two feet of the character in the first motion state. We therefore translate the frame by the difference between the position of the new origin (*i.e.*, the position between the two feet of the character subtracted from an epsilon in Y) and the origin.

Considering that in our experiments, the Kinect faces the person during motion capture, no axis needs to be reversed. On the other hand, a rotation of the frame is necessary. Observing the motion, we can see the character rotated and rising in the air while walking. This is caused by placing the Kinect on a desk (parallel to the floor), it is rotated on the X-axis (so its Y-axis is not normal to the floor). We consider this to be the only rotation to perform, since it is the only one significant enough to be visible when observing the motion. To find the rotation to perform, we therefore consider that the person is facing the Kinect and that they are walking in the direction of the Kinect. For the function representing the position of the character in Y with respect to its position in Z, *i.e.*, its trajectory in 2D on the Y and Z axes, we make a linear trend estimation of the function with the method of least squares in order to obtain a straight line of its trajectory. Taking the difference between the last and the first position could also have been sufficient, but we preferred to have a straight line better representing the reality of the trajectory. Then, we create a trajectory vector which follows the straight line of the trajectory and we apply to the frame the rotation that rotates from the trajectory vector to the new target trajectory (a Z vector).


(a) Before calibration

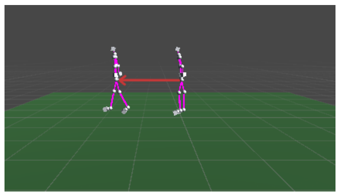

(b) After calibration

Figure 7: Comparison of motions before and after frame calibration

3.3.4 Character building

At this stage, we therefore have the raw motion described by the positions (in a correct frame) of each joint given by the Kinect at each motion state. Now we want to create the physics-based model of the character and describe its motion by the position of its root and the rotations of each joint. For this, we have a description of a standard model containing a hierarchy of articulated rigid bodies that the model must contain, their correspondence with the joints given by the Kinect v2, their mass in proportion to the total mass of the body[2], the primitive which represents them and their width in proportion to their length[1] (to compute the MOI and the collision primitive). We therefore create the physics-based model of the character, from the standard model, the given information of the mass of the person and the length (symmetrized) of each of the articulated rigid bodies. Then, we create the motion of the character, for each motion state, we compute the rotation (cf. Algorithm 1) of each joint relative to the character (cumulative to those of its parents) compared to the state of the standard model (here, a T-pose).

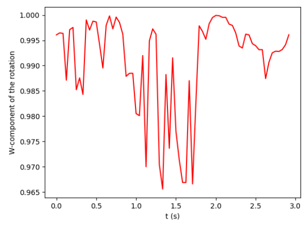
Algorithm 1 Rotation computation algorithm going from one vector to another.

```
Input: $\vec{from} : Vector3, \vec{to} : Vector3$
Output: $Q : Quaternion$
1: $\vec{axis} \leftarrow \vec{from} \times \vec{to}$
2: $angle \leftarrow acos(\hat{from} \cdot \hat{to}) * 0.5$
3: $\vec{axis} \leftarrow \hat{axis} * sin(angle)$
4: $Q \leftarrow cos(angle) + axis_x * \mathbf{i} + axis_y * \mathbf{j} + axis_z * \mathbf{k}$
```

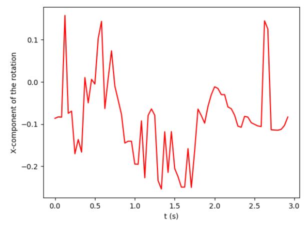
Adapting the model according to the length and the mass of the person will allows us to get correct physics-based simulations and get closer to the best solution (*i.e.*, which will make the optimization phase easier), since the rigid bodies of the character will have the right length, and a mass quite close to reality.

3.3.5 Denoising

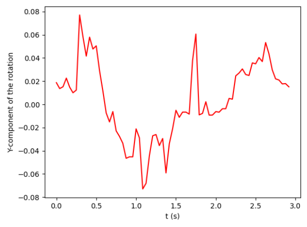
So now we have the physics-based model and motion of our character. But we can see on the visualization of the captured motion that it is very noisy, especially at the feet. We therefore apply a low-pass Butterworth filter with a cutoff frequency of 2Hz (chosen by trial and error, depending on what gave the best results visually on several different motions) on each component of the quaternions of rotations of each joint independently.



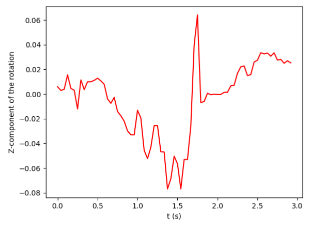
(a) W-component



(b) X-component

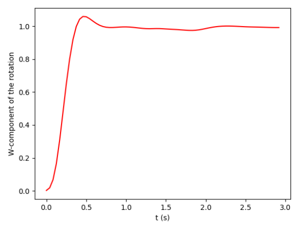


(c) Y-component

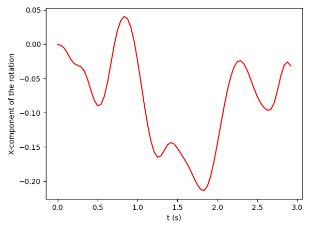


(d) Z-component

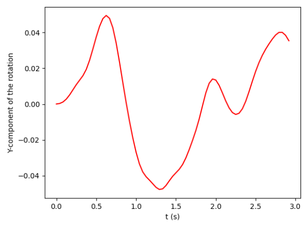
Figure 8: Noised rotation of the right ankle



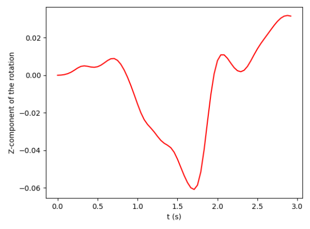
(a) W-component



(b) X-component



(c) Y-component



(d) Z-component

Figure 9: Denoised rotation of the right ankle

3.3.6 Manual finalization

Finally, before converting the motion into input data for the simulator, we do a final post-processing. The simulator needs to know the stance state (*i.e.*, left or right) during the motion. This allows it to adapt in the event that the simulated motion is late or ahead of the transition of the stance. This could have been done during the motion building, by automatically detecting the stance foot (*e.g.*, from the velocity or the position of the feet since the stance foot is supposed to be more or less still and in contact with the ground), but given the low number of motions captured for this preliminary study, we considered it better to do it manually.

So we visualize the motion and we note the transitions of stance, then a small tool that we implemented fills the motion file with the stance state at each motion state. During walking, there is also a double-stance phase, but the SimBiCon controller does not manage this notion. The double-stance phase is just considered to be part of the stance phase that has just begun[4].

3.4 Simulation

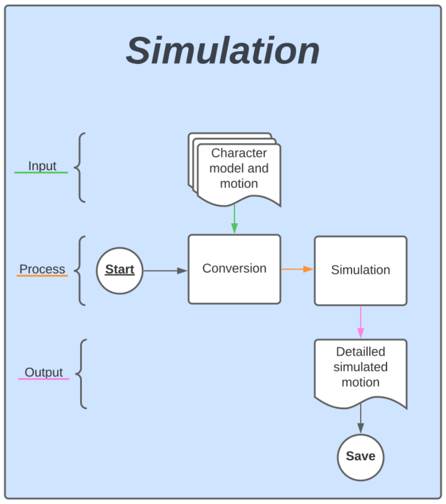


Figure 10: Overview of simulation component

The simulator is already implemented, so we started this part by getting to know the simulator, *i.e.*, learning how to use it (first with their character models and motion data). Next, we have to integrate two processes as shown in Figure 10. A model and motion converter, in order to be able to use the data produced by our motion capture component. As well as an exporter of simulated motions, in order to be able to evaluate the loss function.

3.4.1 Converter

For conversion of the model, a simple transcription of the data is sufficient (*i.e.*, the format remains a hierarchy of articulated rigid bodies with different DOFs, and possessing a mass, a MOI and a collision primitive.

For the motion, more operations are needed. The motion description should be a state machine, where the transition from each state can occur after a fixed duration, or after a new foot contact is established[4]. A state has the rotations of the joints in motion, and the joints are named according to the stance state (*i.e.*, stance or swing), not the side (*i.e.*, right or left). In this implementation, the rotations are given in Euler angles (in radians), and a state can contain a trajectory of rotations instead of a single rotation.

So we start by renaming the joints according to the stance state. Then we convert the rotations of the joints into an Euler angle (which until now was described with a quaternion). The order of the rotation axes can be chosen for each joint, in our case we will describe them all around the Z-axis, then the X-axis, then the Y-axis. Finally, we aggregate all the motion states of the same step, in order to have a state for each step, and to have all our transitions when establishing a new foot contact. We therefore have rotation trajectories at each state.

3.4.2 Export of the simulated motion

Finally, we still have to implement the export of the simulated motion. For this, we interrogate the simulator at a regular interval during the simulation. This interval can be given by the user, and is initially set to $\frac{1}{24}$. However, it cannot be less than the simulator integration time step, which is fixed at $\frac{1}{2400}$ in our implementation (but which can be modified). We want to export all the information that could be used in the loss function. For this, we modified the simulator in order to make certain information accessible.

In the description of the simulated motion, we have the information of the character state: its position, its rotation, its velocity, its angular velocity, its heading, then for each of its joints: their rotation and their angular velocity. As well as the information of the controller state: the stance state, the torques applied to each joint, if a rigid body of the character is in contact with the ground, and the desired pose (containing all the information of a character state mentioned above).

All this information is accessible from the simulator, simply by creating public accessors. From this it is possible to compute some other information that might be useful. For example, we can compute the position of each joint, given their rotation (and that of their parents), and then compute their velocity. Also, the information of the contact points of the rigid bodies in the world is accessible, containing for each contact: the two rigid bodies in collision, the force applied to the first rigid body (therefore also the negative force applied to the second), the position of the point of contact and its normal, as well as the penetration depth. From this information, we can deduce other information that can be integrated into the loss function. For example, the stance state in the controller does not really correspond to which foot is in contact with the ground (there is no double-stance phase), so one could infer which feet are in contact with the ground. We haven't implemented any of these methods, but we may come back to them later depending on the choice of the loss function.

3.5 Model optimization

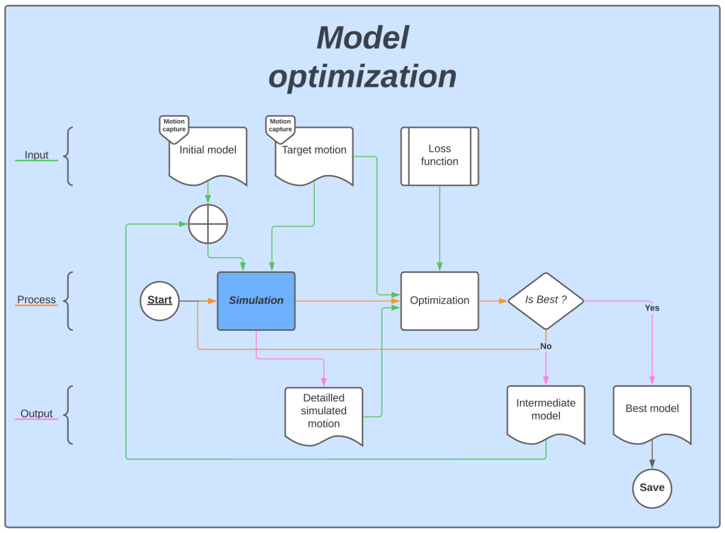


Figure 11: Overview of model optimization component

The optimization of the model is at the center of this work, since the objective of the internship is its design, and it constitutes the main scientific obstacle (particularly for the design of the loss function). The bibliographic study was therefore very important for this component.

As shown in the Figure 11, from a character model and its motion provided by the motion capture component, the optimization produces the model that best satisfy the target motion. For this, at each step of the optimization, for each intermediate model (*i.e.*, model modified during the optimization), it simulates the motion captured as a target motion. Then, from the target motion and the simulated motion, a loss function is evaluated in order to know the modifications to be made to the new intermediate model of the next step, until finding the best model.

3.5.1 Parameters

The parameters of the model to be optimized are: the mass and the 3 components of the MOI (independently) of each rigid body. Since the lengths of the rigid bodies are those of the real person, and the collision primitives don't need to be precise in our study, since there isn't really supposed to be any collision apart from the feet with the ground.

More precisely, the parameters to be optimized will be normalizations of these components (*i.e.*, ranging from 0 to 1), in order to make easier the optimization. For normalization, we need to assign a minimum value and a maximum value for each component. We choose them according to the expected results.

For the mass of each rigid body, we start by looking at the standard deviation of the mass of a body segment proportional to its average mass (for the Korean population[2]), *i.e.*, $\frac{\sigma}{\bar{x}}$ in which σ is the standard deviation of the mass and $\bar{x}$ is the average mass. Then we consider by intuition that the mass of rigid bodies should not deviate by more than $\frac{2}{3}$ from this value compared to its value of the initial model x_0 already adapted to the mass of the person, *i.e.*, $x_0 * \frac{2*\sigma}{3*\bar{x}}$. Then, for this preliminary study, since we want first of all check the consistency of the model optimization, we add a mass w to the upper bound, for the body segments on which we can hang a weight for the imitation of pathological motion. In which w is the mass of the hanged weight (*i.e.*, 9.5 for trunk, 2.3 for ankles and 1.5 for wrists). We thus finally have a bound of $\left[x_0 - (x_0 * \frac{2*\sigma}{3*\bar{x}}); x_0 + (x_0 * \frac{2*\sigma}{3*\bar{x}}) + w\right]$ for the mass of rigid bodies.

For the MOI, we can consider that the mass of each rigid body should be more or less uniformly distributed for each dimension, *i.e.*, the expected result is close to the initial value x_0. We can therefore simply set a small coefficient c (*e.g.*, $c = 0.1$), and set the bound of each MOI's component by $\left[x_0 - (x_0 * c); x_0 + (x_0 * c)\right]$.

3.5.2 Strategy

Then, for the choice of the optimization strategy, we use Covariance Matrix Adaptation (CMA)[3], since it is widely used in the field of physics-based character animation (*e.g.*, to optimize model or controller parameters[17][11]).

We choose the initial standard deviation σ_0 of the CMA optimizer according to the expected results. The smallest expected deviations should be around 0.1 and the largest should be around 0.8 (for the case of the hand with the weight hangged to the wrist). We need to choose our σ_0 so that the optimum results are around $x_0 \pm 3 * \sigma_0$. In our case, the expected deviation of the optimum from the initial value is not the same for each parameter. We therefore take the smallest deviation into account, and choose $\sigma_0 = 0.03$.

3.6 Loss function

For the design of our loss function, we must think about its form and its composition according to our study of the existing works and what makes sense within our work. Then, we will adjust the variables during the experiments on several captured motions (healthy and pathological). Then, we will interpret these results by comparing to the expected results (*i.e.*, to the differences of models corresponding to the real weights hanged to the body).

Our loss function is the function that the optimizer must minimize. It will compare the simulated motion and the target motion and return a loss value.

For its design, we will not limit ourselves to what is currently implemented, but to what is possible to implement with our simulator (*e.g.*, those described in Section 3.4.2).

3.6.1 Termination conditions

In the context of optimization using simulation, the termination condition [12][16][18][19] is a condition for which, if true during the simulation, the simulation is prematurely terminated, and the loss function returns a very large loss for this simulation. The very large loss attributed helps to avoid local minima in the optimization, and the premature termination of the simulation saves time, since it is known that this simulation is not valid, so there is no need to go to the end of it.

In our case, the simulation is not done during the loss function, so it is not possible to stop the simulation prematurely. But we still use the notion of invalid results, producing a very large error (namely, 100 in our case). If our condition is verified, the loss function is directly returned.

We take a single termination condition: if the character falls, the simulation is invalid. To detect the fall, we verify if :

$$\exists_{t \in [0;t_{max}]}, h_{COM}(t) < 0.1 * h_{COM}(0) \tag{13}$$

in which t_{max} is the duration of the motion, $h_{COM}(t)$ is the height (*i.e.*, the Y component of the position) of the character in the simulated motion at a given instant t and therefore, $h_{COM}(0)$ is the height of the character in the initial state.

3.6.2 Form

Before specifying the components of the loss function, we can describe its general form.

The loss to be minimized $\bar{E}$ is defined as:

$$\bar{E} = \sum_{m \in s} \bar{E}_m \tag{14}$$

in which s is the set of measurement components of loss.

A loss measure $\bar{E}_m$ is defined as:

$$\bar{E}_m = W_m * \left\{ \int_0^{t_{max}} 1 - exp(-c_m * \|E_m(t)\|)\delta t \right\}_{H_m} \tag{15}$$

in which t_{max} is the motion duration, W_m is a weight, and $\{\}_{H_m}$ enforces a threshold, *i.e.*, the value between braces is set to zero if it is lower than a margin of error H_m.

An error measure $E_m(t)$ returns an error value in $\mathbb{R}^{n \in \mathbb{N}^*}$ for an instant t. The error measurements of each component will be detailed in Section 3.6.3.

The expression $\hat{E}_m(t) = 1 - exp(-c_m * \|E_m(t)\|)$ therefore corresponds to the normalized error measure, where c_m is an adjustment coefficient. It is used to scale the error measurements $E_m(t)$. It will be chosen according to what is considered to be a small or a large error.

For example, by comparing two walking motions, we can intuitively say that their position is very similar if there is $10cm$ of distance between their COM, and conversely, a distance of $1m$ will seem large. So, for a positioning error measure E_{pos} (which we will detail later), we can initialize $c_{pos} = 2$. So we will have $\hat{E}_{pos}(t) \approx 0.18$ (small error) for $\|E_{pos}(t)\| = 0.1$, and $\hat{E}_{pos}(t) \approx 0.86$ (large error) for $\|E_{pos}(t)\| = 1$.

3.6.3 Terms of the function

Character positioning

$$E_{pos}(t) = p_{COM}(t) - \tilde{p}_{COM} \tag{16}$$

in which $p_{COM}(t)$ is the position of the character's COM at an instant t in the simulated motion, and $\tilde{p}_{COM}(t)$ that of the target motion.

This error measure encourages the character's center-of-mass position to follow that of the target motion.

Character velocity

$$E_{vel}(t) = v_{COM}(t) - \tilde{v}_{COM}(t) \tag{17}$$

in which $v_{COM}(t)$ is the velocity of the character's COM at an instant t in the simulated motion, and $\tilde{v}_{COM}(t)$ that of the target motion.

This error measurement penalizes deviations in the character's velocity from that of the target motion.

Character heading

$$E_{head}(t) = \psi_{head}(t) - \tilde{\psi}_{head}(t) \tag{18}$$

in which $\psi_{head}(t)$ is the angle of the character's heading at an instant t in the simulated motion, and $\tilde{\psi}_{head}(t)$ that of the target motion.

This error measurement penalizes deviations in the character's heading from that of the target motion.

Rigid bodies positioning

$$E_{local}(t) = \frac{\sum_{r \in S_{rigid}} p_r^{local}(t) - \tilde{p}_r^{local}(t)}{|S_{rigid}|} \tag{19}$$

in which S_{rigid} is the set of rigid bodies, $|S_{rigid}|$ its cardinality (*i.e.*, the number of rigid bodies), $p_r^{local}(t)$ the position of the rigid body r in the character's frame at an instant t in the simulated motion, and $\tilde{p}_r^{local}(t)$ that of the target motion.

This error measure encourages the character's pose to match the positions of the target motion.

Joint angular velocities

$$E_{angvel}(t) = \frac{\sum_{j \in S_{joint}} \omega_j(t) - \tilde{\omega}_j(t)}{|S_{joint}|} \tag{20}$$

in which S_{joint} is the set of joints, $|S_{joint}|$ its cardinality (*i.e.*, the number of joints), $\omega_j(t)$ the angular velocity of the joint j at an instant t in the simulated motion, and $\tilde{\omega}_j(t)$ that of the target motion.

This error measurement penalizes deviations in the joints's angular velocity from that of the target motion.

End-effectors positioning

$$E_{end}(t) = \sum_{e \in S_{end}} p_e^{local}(t) - \tilde{p}_e^{local}(t) \tag{21}$$

in which S_{end} is the set of end-effectors (*i.e.*, $S_{end} = [leftfoot, rightfoot, lefthand, righthand]$), $p_e^{local}(t)$ the position of the end-effector e in the character's frame at an instant t in the simulated motion, and $\tilde{p}_e^{local}(t)$ that of the target motion.

This error measurement encourages the character's hands and feet to match the positions from the target motion. The positioning error of the end-effectors is already taken into account in E_{local}, but adding another component makes it possible to assign variables specific to this error measurement (*i.e.*, W_{end} , H_{end} and c_{end}).

Effort

$$E_{effort}(t) = \frac{\sum_{j \in S_{joint}} \mathcal{T}_j(t)}{m * |S_{joint}|} \tag{22}$$

in which S_{joint} is the set of joints, $|S_{joint}|$ its cardinality (*i.e.*, the number of joints), m the mass of the character, and $\mathcal{T}_j(t)$ the torque applied to the joint j at an instant t in the simulated motion.

This error measurement encourages effort minization.

Foot sliding

$$E_{slide}(t) = c^{ground}_{lfoot}(t) * v_{lfoot}(t) + c^{ground}_{rfoot}(t) * v_{rfoot}(t) \tag{23}$$

in which $c^{ground}_x(t) = \begin{cases} 1, & \text{if } x \text{ is in contact with the ground at an instant } t \\ 0, & \text{otherwise} \end{cases}$ in the simulated motion and $v_x(t)$ the velocity of x at an instant t in the simulated motion.

This error measurement prevents to end up with an incorrect motion caused by a foot sliding by penalizing through contact foots velocity.

Skipping

$$E_{skip}(t) = \neg(c^{ground}_{lfoot}(t) + c^{ground}_{rfoot}(t)) \tag{24}$$

in which $c^{ground}_x(t) = \begin{cases} 1, & \text{if } x \text{ is in contact with the ground at an instant } t \\ 0, & \text{otherwise} \end{cases}$ in the simulated motion and $\neg(x) = \begin{cases} 1, & \text{if } x = 0 \\ 0, & \text{otherwise} \end{cases}$.

This error measurement prevents to end up with a skipping motion instead of a walking motion by penalizing through non-contact of both feet. Since we are focusing on walking motions, so there is not supposed to be a phase where neither foot is in contact with the ground.

3.7 Entire expression and variables value

We thus finally have the loss function to minimize defined as :

$$\bar{E} = \sum_{m \in s} W_m * \left\{ \int_0^{t_{max}} 1 - exp(-c_m * \|E_m(t)\|) \delta t \right\}_{H_m} \tag{25}$$

in which $s = \{pos, vel, head, local, angvel, end, effort, slide, skip\}$, t_{max} is the motion duration, and the variables are those described in the Table 1.

	$\bar{E}_{pos}$	$\bar{E}_{vel}$	$\bar{E}_{head}$	$\bar{E}_{local}$	$\bar{E}_{angvel}$	$\bar{E}_{end}$	$\bar{E}_{effort}$	$\bar{E}_{slide}$	$\bar{E}_{skip}$
W_m	10	10	50	10	10	15	1	30	20
H_m	0.1	0.2	0.35	0.2	0.3	0.2	0	0.1	0
c_m	2	5	1.5	10	15	10	0.5	15	40

Table 1: Variables value for the individual error measures

3.8 Experimentation and troubles

To be able to experiment, our working environment must be fully functional. In our case, everything is functional except for converting motion into input data for the simulator described in Section 3.4.1. No experimentation could therefore be carried out, and no experimental result concerning the computation method of the model could emerge from this work.

To clarify the problem, everything described works, the simulator accepts the input data and starts the simulation, but the character instantly loses balance when it shouldn't. Adding only the captured motion of the shoulders to a motion provided in the "Cartwheel" project, we can see the character flapping his arms, which is not supposed to happen at all. However, converting rotations to Euler angles has been checked several times, it is correct, and in the case of this test, the given shoulder motion is correct in every point. Currently, none of my knowledge of the simulator allows me to understand where could come from the problem.

4 Conclusion

The objective of the internship was the design of the model optimization, but finally, I spent almost all my internship on the implementation of the work environment, including a very large part of this time on troubles caused by my misunderstanding of how the simulator works. Until the end, this is what disallows experimentation. No conclusion can therefore be drawn from this work concerning the computation method of the model.

This work can nevertheless be useful for later work. It contains my initial thoughts on the design of model optimization from captured motion, a list of simulation information that may be useful for its loss function. Also, for more general work, a list of post-processing for captured motions are described here.

References

[1] Claire C. Gordon, Thomas Churchill, Charles E. Clauser, Bruce Bradtmiller, John T. McConville, Ilse Tebbetts and Robert A. Walker. 1989. "Anthropo-

metric Survey of U.S. Army Personnel: Summary Statistics, Interim Report for 1988"

[2] Se Jin Park, Chae-Bogk Kim and Soo Chan Park. 1999. "Anthropometric and Biomechanical Characteristics on Body Segments of Koreans"

[3] Nikolaus Hansen. 2006. "The CMA Evolution Strategy: A Comparing Review"

[4] KangKang Yin, Kevin Loken and Michiel van de Panne. 2007. "SIMBICON: Simple Biped Locomotion Control."

[5] Hartmut Geyer and Hugh Herr. 2010. "A Muscle-Reflex Model That Encodes Principles of Legged Mechanics Produces Human Walking Dynamics and Muscle Activities"

[6] Benjamin J. Stephens and Christopher G. Atkeson. 2010. "Dynamic Balance Force Control for Compliant Humanoid Robots"

[7] de Lasa, M., Mordatch, I., Hertzmann, A. 2010. "Feature-Based Locomotion Controllers."

[8] Stelian Coros, Philippe Beaudoin and Michiel van de Panne. 2010. "Generalized Biped Walking Control."

[9] Jack M. Wang, David J. Fleet and Aaron Hertzmann. 2010. "Optimizing Walking Controllers for Uncertain Inputs and Environments"

[10] Jamie Shotton et al. 2011. "Real-Time Human Pose Recognition in Parts from Single Depth Images"

[11] Wang, J., Hamner, S., Delp, S., Koltun, V. 2012. "Optimizing Locomotion Controllers Using BiologicallyBased Actuators and Objectives."

[12] Thomas Geijtenbeek and Michiel van de Panne and A. Frank van der Stappen. 2013. "Flexible Muscle-Based Locomotion for Bipedal Creatures."

[13] E. Lachat, H. Macher, M.-A. Mittet, T. Landes, P. Grussenmeyer. 2015. "First Experences with Kinect v2 Sensor"

[14] Wenbing ZHAO. 2016. "A concise tutorial on human motion tracking and recognition with Microsoft Kinect"

[15] John Schulman et al. 2017. "Proximal Policy Optimization Algorithms"

[16] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. "DeepMimic: Example-Guided Deep Reinforcement Learning of PhysicsBased Character Skills."

[17] Nuttapong Chentanez, Matthias Müller, Miles Macklin, Viktor Makoviychuk, and Stefan Jeschke. 2018. "Physics-based Motion Capture Imitation with Deep Reinforcement Learning"

[18] Soohwan Park, Hoseok Ryu, Seyoung Lee, Sunmin Lee, and Jehee Lee. 2019. “Learning predict-and-simulate policies from unorganized human motion data.”

[19] Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. 2019. “DReCon: Data-Driven Responsive Control of Physics-Based Characters.”

[20] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. 2021. “AMP: Adversarial Motion Priors for Stylized Physics-Based Character Control.”