Master of Engineering Thesis

A physics-based Juggling Simulation using Reinforcement Learning

February 2019

Seoul National University
Department of Computer Science and Engineering
Jason Chemin

A physics-based Juggling Simulation using Reinforcement Learning

This paper was submitted as a Thesis for Master Degree in Engineering

November 2018

Seoul National University

Department of Computer Science and Engineering

Jason Chemin

Confirmation of Jason Chemin's Master Thesis

February 2019

Chairman	신영길	(인)
Vice Chairman	이제희	(인)
Commitee	서진욱	(인)

Abstract

Juggling is a physical skill which consists in keeping one or several objects in continuous motion in the air by tossing and catching it. Jugglers need a high dexterity to control their throws and catches which require speed, accuracy and synchronization. Also, the more balls we juggle with, the more those qualities have to be strong to achieve this performance. This thesis follows a previous project made by Lee et al.[1] where they performed juggling to demonstrate their method. In this work, we want to generalize the juggling skill and create a real time simulation by using machine learning. A reason to choose this skill is that ”Studying the ability to toss and catch balls and rings provides insight into human coordination, robotics and mathematics” as written in the article Science of Juggling[2]. That is why juggling can be a good challenge for realistic physical based simulation to improve our knowledge on these fields, but also to help jugglers to evaluate the feasibility of their tricks. In order to do it, we have to understand all the different notations used in juggling and to apply the mathematical theory of juggling to reproduce it.

In this thesis, we find an approach to learn juggling. We first break the need of synchronization of both hands by dividing our character in two.

Then we divide the juggling into two subtasks catching and throwing a ball, where we present a deep reinforcement learning method for both of them. Finally, we use these tasks sequentially on both sides of the body to recreate the all juggling process.

As a result, our character learns to catch all balls randomly thrown to him and to throw it at the velocity wanted. After combination of both subtasks, our juggler is able to react accurately and with enough speed and power to juggle up to 6 balls, even with external forces applied on it.

Keywords : Juggling, Physics-based character animation, Reinforcement Learning

Student Number : 2017-22169

Contents

List of figures

List of tables

Chapter 1

Introduction

The art of juggling has appeared in various cultures in the history as a distracting skill which can sometime be very impressive as told in some ancient Chinese annals, a man who juggled nine balls in front of the enemy armies and won by impressing them with its dexterity. The juggler has to show his control over the timing of his tosses and catches if he wants to keep all objects in the air. Specifically, throwing a ball is a task requiring a balance between accuracy to avoid collision and power to throw the ball high enough. And the more balls we juggle with, the higher or faster we have to throw the ball while keeping track of others in the air. The actual human world record of juggling continuously is eleven balls and as explained by the professional juggler Jack Kalvan [3], this is physically very difficult to beat.

Research on juggling started with Shannon [4] when he formulated a mathematics juggling theorem and built the first machine which was able to perform bounce juggling with three balls. Since then, this physical skill has been an interesting challenge in science and in robotics [2]. The first juggling machines were designed to make the balls follow a predefined trajectories as the Shannon's juggling machine which was built with no feedback, repeating the same movement with a constant frequency and avoiding collision by the design of its system. The trajectory of the ball was corrected using groove cups or tracks to redirect it. Many juggling machines were conceived this way, but it is only recently that we started to use feedbacks from cameras and other sensors to correct trajectory errors while juggling and to adapt their behavior. The machine named ServoJuggler [5] can juggle with 5 balls using visual feedback and sensors to detect catches. The humanoid robot created by Kober et al. [6] is able to juggle with a a human partner using one hand. It uses external cameras to locate the ball and to predict its trajectory. Then it throws it back at the right velocity to the other human participant.

In computer graphics field, a physics-based simulation could help jugglers to create, visualize and evaluate the feasibility of patterns. Some methods can be applied to calculate the movements needed to throw a ball at the right velocity and the position where the ball can be catch, but they can be difficult to apply for real-time. The interactive tool of Jain et al. [7] is used to manually edit motion to create dynamic scenes with human and object interaction as toss or football juggling.

This project follows a previous project realized by Lee et al. [1] where they use a framework for simulation and control of a human musculo-skeletal system. In order to demonstrate the efficacy of their model, they perform a very realistic juggling where a character is able to juggle up to 6 balls alone and more with two persons juggling together. The Finite Element Method used to calculate the action of the agent needs a pre-processing of several seconds per frame to create the simulation.

The recent research on machine learning provides more possibilities to solve the real-time problem. To demonstrate their method to learn a feedback policy, Ding et al. [8] learns to keep a ball in the air using a racket and to kick a ball to reach a target. Reinforcement learning combined with neural networks has led to many successes in learning policies to perform complex tasks as playing Atari games [9]. Applied to physically based simulation an agent can learn locomotion behavior in rich environment using Proximal Policy Optimization algorithm [10]. Using reinforcement learning to imitate actions in motion clips, Peng et al. [11] make an agent learn to throw a ball on a target like a Baseball pitch, reproducing human movement from motion capture. Their character is able to throw a ball using the whole body to reach a target.

In this project, we would like to focus more on the Juggling Skill himself and all its possibilities. Our goal is to create an agent on a 2D simulated body and 3D later on, able to learn how to juggle in real time using Deep Reinforcement Learning. Juggling is composed of two main subtasks which are catching and throwing a ball. We train two agents to perform these subtasks with the right arm only. Finally, the trained agents can be mirrored on the left arm and used sequentially to recreate the all process of juggling.

Chapter 2

Juggling theory

2.1 Notation and Parameters

Toss juggling can be resume as a sequence of throws and catches from one or two hands at different heights, timings, directions and powers. Therefore, to describe the figures performed and the future sequence of catches and throws, a codification is needed. The most common notation used is named Siteswap. It is a tool to help jugglers to communicate with each other about their tricks and to describe, to share and to discover new patterns and transitions. Some extensions of this notation can describe very complex patterns. In this work, we will use only the basic notation named vanilla Siteswap which describes patterns juggled by one person and where one object is caught and thrown at a time. As shown in Figure 1, at every periodic beat time, a throw is executed from one hand alternating between the right and left hand. The hand period corresponds to the time between two throws from the same hand and the dwell time is the duration of the hand holding a ball during this period. The dwell time can also be seen as the time available to prepare the throw. Regarding the measurements made on several professional jugglers [3], the hand period P_{hand} is around 0.55 seconds with three balls down to 0.44 seconds when juggling with seven balls . During this period, the performer has to catch the ball incoming and then

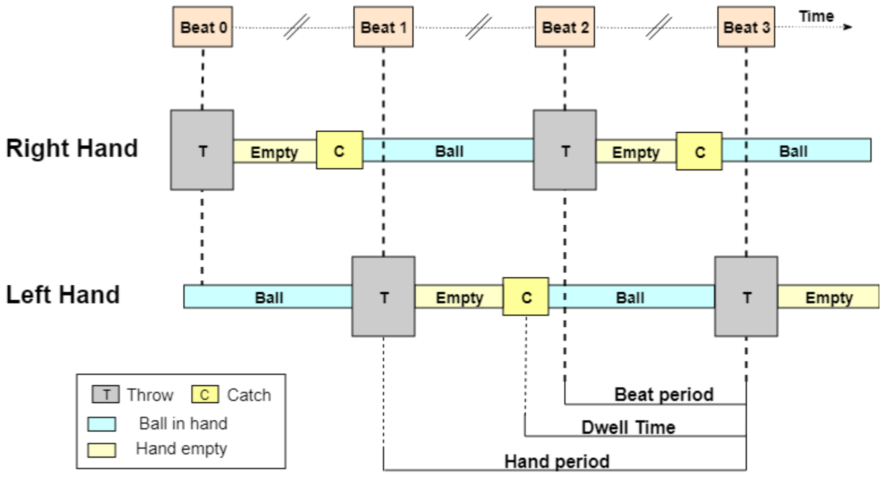


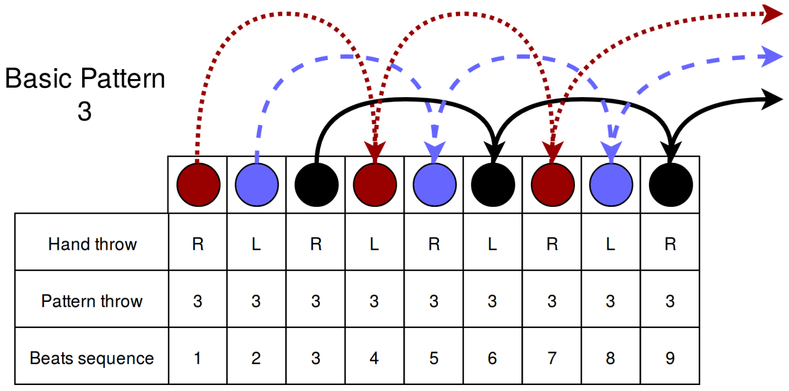
Figure 1: Asynchronous juggling system. One ball is thrown (T) from the right hand at every even beat, and one ball is thrown from the left hand at every odd beat. After a throw, our hand is empty and a new ball is caught (C).

needs some time to prepare the throw. The dwell time corresponds to the hand period multiplied by the dwell ratio D_{ratio} which is in average around 70%.

To understand the juggling process, we will describe the actions of the left hand in Figure 1. At the beginning, the left hand has a ball and is preparing a throw. Then, the beat 1 comes and we throw this ball (T). After the throw, our left hand is empty until we catch a new ball (C). After the catch, we have a ball in hand and we prepare the next throw at beat 3. In this example, the odd beats correspond to a throw from the left hand, and the even beats, a throw from the right hand. We do not specify where the ball should be thrown, and this is explained by the basic patterns.

2.2 Juggling patterns

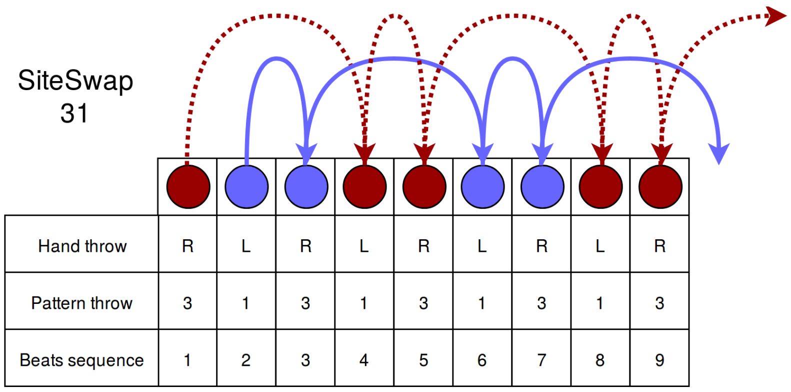
We name basic pattern or basic Siteswap, the simplest kind of juggling patterns and its number corresponds to the number of beats you have to wait before throwing this ball again. We can see an example on Figure 2 with Basic a Pattern 3 performed, where The ball caught at the first beat will have to be thrown again 3 beats later. This means that the ball will have to be caught at the hand period just before this beat. As the hand who performs the throw alternates between right and left at every beat, an odd basic pattern represents a throw that will go from one hand to another, and an even one is a throw for the same hand.



Hand throw	R	L	R	L	R	L	R	L	R
Pattern throw	3	3	3	3	3	3	3	3	3
Beats sequence	1	2	3	4	5	6	7	8	9

Figure 2: Sequence of a basic pattern (3).

A vanilla Siteswap is a cyclic sequence of several basic patterns where we will throw a ball at each beat with the next pattern in this sequence. It exists as many vanilla Siteswap patterns as there are combinations possible where we have only one catch per hand period and one ball thrown at each beat. In this paper, we will name Siteswap only sequences of at least two different basic patterns. On figure 3 is an example of a Siteswap ”31” where the first ball thrown will have to be thrown again 3 beats later, and the second ball will have to be thrown again 1 beat later. Then we repeat this sequence of Basic Patterns for all future throws.



Hand throw	R	L	R	L	R	L	R	L	R
Pattern throw	3	1	3	1	3	1	3	1	3
Beats sequence	1	2	3	4	5	6	7	8	9

Figure 3: Sequence of a Siteswap (31).

By juggling with only basic patterns, the number of ball we can have is equal to the pattern used. For example, juggling with a basic pattern ’4’ allows you to juggle with four balls and so on. For a Siteswap, the number of balls we juggle with is equal to the sum of all basic patterns of the sequence divided by its length. For example, a Siteswap ’531’ will be juggled with three balls.

We associate a throw to a basic pattern. The higher the basic pattern is, the more the ball will have to stay in the air. And this means that the throws will have to be more powerful to make the balls go higher. The mathematics of juggling give us a way to calculate the time the ball will stay in the air according to the pattern performed and some juggling parameters :

$$T_{ballInAir} = P_{hand}(0.5 * C_{pattern} - D_{ratio}) \tag{2.1}$$

where $T_{ballInAir}$ is the time the ball will have to spend in the air before being caught, $Pattern$ is the basic pattern of this throw, P_{hand} is the hand period, D_{ratio} the Dwell time divided by the hand period, in other words the percentage of time the ball will stay in the hand during this period. We set these values to 0.55 seconds and 70 percent as said in the previous section. Finally $C_{pattern}$ is the basic pattern of the throw which needs to be superior or equal to two. This is because a Basic Pattern 1 corresponds to a very fast throw where the ball will need to be thrown again at the next beat. This requires to set a lower value for the dwell ratio that we set to 0.20 percents for the basic pattern 1.

Knowing the time the ball will stay in the air and other juggling parameters, we can calculate the velocity needed for each pattern :

$$V_{yThrow} = 9.81(T_{ballInAir}/2.0) \tag{2.2}$$

$$V_{xThrow} = \frac{D_{Catch,Throw}}{T_{ballInAir}} \tag{2.3}$$

where V_{yThrow} and V_{xThrow} is the velocity wanted for the throw on the Y-axis and X-axis and $D_{Catch,Throw}$ is the distance between the throw and the catch at the same height. Performing a throw of a specific basic pattern consists in tossing the ball at the velocity V_{xThrow} and V_{yThrow} associated to it.

Chapter 3

Approach to learn juggling

The overall juggling process can be explained as throwing the ball at the right velocity and time, and catching the next ball simultaneously with our both hands. A human has to think about both tasks at the same time and this can make the learning of juggling very difficult for him. That is why in our simulation, we want to break this need of synchronization by dividing our character in two parts, a right and left side independent from each other. Then, we want to learn separately the two different tasks catching and throwing using Deep Reinforcement Learning (Deep RL). We first train our agents only on the right arm of our body and we mirror it on the left arm. Then, in parallel on both sides of the body, we use our catching and throwing agent sequentially to recreate the all process of juggling.

3.1 Juggling sequence

During the simulation, we keep track of the beat time for the left and right hand. At every even beat, we throw a ball from the right hand and at every odd beat, from the left. The switching between the catching and throwing agent for one arm is straightforward as shown in Figure 4. When we have a ball in hand, we are in throwing mode and when our hand is empty, we are in catching mode. Therefore, we need to detect the frame

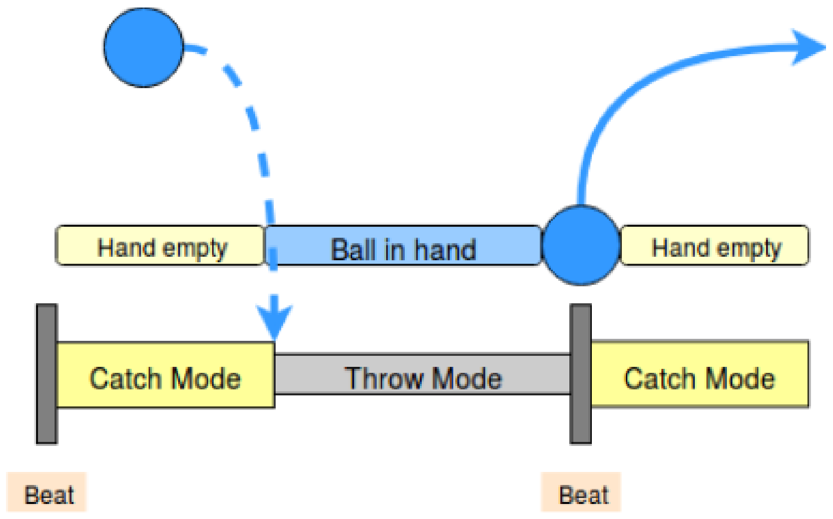


Figure 4: Sequence of catch and throw mode for one hand.

when the ball is caught and when the ball is thrown. Also, the catching task needs to adapt to possible external forces that could be applied on balls while in the air or to a late catch which could lead to a very reduced Dwell time and so less time to prepare the throw. As a consequence, the throwing agent will have to adapt to succeed in throwing even in this difficult situation.

3.2 Reinforcement learning

Recently, Reinforcement Learning (RL) has shown a lot of success in physics-based simulation creating robust agents able to perform tasks as walking, running or imitating performances from Motion Capture data. We will explain how it works and focus on one RL algorithm named Deep Deterministic Policy Gradient that can be use to learn our both subtasks catching and throwing a ball.

3.2.1 Definition

Reinforcement learning is defined as a Markov decision process $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ a set of actions discrete or continuous, $\mathcal{P}(s, a, s')$ is the transition probability $p(s'|s,a)$, $\mathcal{R}$ a reward function mapping $\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$, and $\gamma \in [0,1)$ a discount factor.

We train an agent to take some action in a fully observable environment, which returns him a reward to encourage him or not to do this action next time. The goal is to learn a policy which gives the action to perform in function of the actual state $a_t = \pi(s_t)$ in order respect the Bellman optimality equation, in other words to maximize the future cumulative rewards named Q-value.

$$Q(s,a) = \mathcal{R}(s,a) + \gamma \max_{a' \in \mathcal{A}} Q(s', \pi(s')) \tag{3.1}$$

In a discrete state action space, calculating the Q-value is straightforward as we can compute all the rewards we get at each state from the goal and set this future expected reward in output for each action available. Then we select the best action to perform maximizing this Q-value. But in continuous space, we have to find other methods to calculate this value as there are an unlimited number of actions possible. A solution could be to discretize the space to get a finite number of actions to perform. If we discretize the space too much, we can get a big number of action. This will increase the size and number of parameters of our network and so increase its learning and running time. On the other hand, if we perform a very small discretization of the space, the number of available actions may not be enough to perform what we want in our simulation.

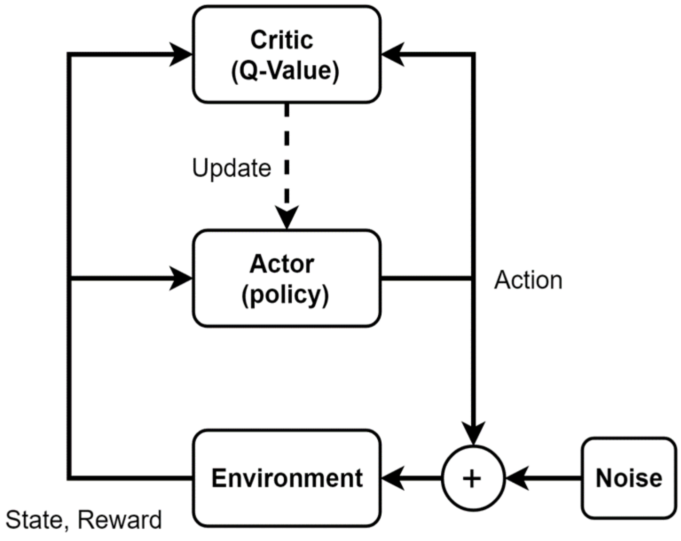


Figure 5: Deep Deterministic Policy Gradient model.

That is why for this simulation we choose to use Deep Deterministic policy gradient (DDPG) [12] which is a model-free RL algorithm used for actions in continuous space. It is composed of two parts, a Critic and an Actor as shown on Figure 5. The goal of the Critic is to learn the future cumulative reward Q-value we can get by being in one state and taking one action. Then it updates the Actor, containing the Policy, toward the best action to perform.

3.2.2 Advantages

Reinforcement Learning major advantage is that it can learn by itself. We have to find a balance between Exploration of new actions to perform when encountering some situations, and Exploitation of what we learnt in order to improve our skill in this task. We have to go for Exploration at the early phase of the training to learn as much as possible by updating our policy. And finally the more we learn, the more we will go for exploitation.

A second advantage is that it uses Neural Networks for its policy. In the case of the Actor Network in DDPG, the time to query an action is very fast and suits well for real time animation. In our final result, the time to query one action for one arm was inferior to 0.5ms. Considering that our simulation timestep is around 10ms, calculating the action of both arms takes approximately 1ms which is inferior to our simulation time step and allows us to have a real time simulation.

The final advantage of Reinforcement Learning in physics-based simulation is that we can get robust agents able to perform their task even under difficult situations as pushing a character while he is walking. In our case we want our character to keep on Juggling with wind added on balls while in the air.

After choosing the Reinforcement Learning algorithm used, what we need to do is to design our model. At first, we set the parameters of our Actor-Critic Networks which can be seen on Table 1. Then we need to design the reward for each task to evaluate the quality of what he is doing. After that, we need to define the observable states of our agent, what we need to know of the environment in order to act. And finally, we define the action-space.

3.3 Rewards for Juggling

3.3.1 Catching

In this simulation, we consider that the ball is automatically caught by the hand if their positions are close. So the goal of the catching task is to put the hand on the ball and if possible, near the height of catch wanted. The catching reward is composed of two terms :

$$R = w_1 R_{effort} + w_2 R_{catch} \tag{3.2}$$

where the weights w_1 and w_2 are constant values. The first term, R_{effort} encourages the agent to move smoothly by punishing jerks.

$$R_{effort} = -\sum_{j \in \mathcal{J}} |\omega_t^j - \omega_{t-1}^j| \tag{3.3}$$

where $\mathcal{J}$ is a set of joints, ω_t^j and ω_{t-1}^j are the angular joint velocity at time t and $t-1$.

The second term $R_{catching}$ penalizes the agent for not catching a ball or catching it far from the height wanted that we name in this work, catching line.

$$R_{catching} = \begin{cases} -1, & \text{if fail to catch the ball} \\ -|\bar{p}_y - p_y| & \in \text{(-1., 0.] otherwise} \end{cases} \tag{3.4}$$

where $\bar{p}_y$ and p_y are the height of catch wanted and the actual height of catch respectively.

The first term is applied at every step whereas the second one is applied only once per episode at the terminal state. The episode ends when the ball is caught or lands out of the action range of the arm.

3.3.2 Throwing

To simplify the throwing task, we decide to release the ball automatically, respectively to the Juggling beat times. In this condition, the goal of throwing is at the release time to move our hand with the grasped object near the release position wanted at the right velocity. The throwing reward is composed of three terms :

$$R = w_3 R_{effort} + w_4 R_{positionThrow} + w_5 R_{velocity} \tag{3.5}$$

where the weights w_3, w_4 and w_5 are constant values. The first term, R_{effort} encourages the agent to move smoothly by punishing jerks.

$$R_{effort} = -\sum_{j \in \mathcal{J}} |\omega_t^j - \omega_{t-1}^j| \tag{3.6}$$

The second term, $R_{positionThrow}$ penalizes the agent for throwing the ball far from the release point wanted :

$$R_{positionThrow} = \begin{cases} - \parallel \bar{p} - p \parallel^2 & \text{when ball thrown} \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

where $\bar{p}$ is the position where the hand should release the ball and p is the position of the actual release point. The last term $R_{velocity}$ penalizes the agent for not throwing the ball at the velocity wanted :

$$R_{velocity} = \begin{cases} - \parallel \bar{v}_{throw} - v_{throw} \parallel^2 & \text{when ball thrown} \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

where $\bar{v}_{throw}$ and v_{throw} are the velocity of the ball wanted and actual respectively when thrown.

The first term is applied at every step whereas the second and third terms are applied only once per episode at the terminal state. The episode ends when the ball is thrown.

Chapter 4

Experiments and Results

We first test each agent for what they have been trained and we finally use them sequentially to create our 2D simulated juggler. As a result, the combination of both agents on our character is able to perform Siteswap composed of basic patterns up to 6 and we test the robustness of our agent by adding some wind on balls. Finally we make him start juggling with 2 balls, and at the same time throw him some more balls gradually up to 6.

4.1 Experiments

Our algorithm was implemented in Python, using DART for the simulation of articulated body dynamics and TensorFlow for the training of deep neural networks. The simulation was run on a PC with Intel Core i7-4930K (6 cores, 3.40GHz). Parameters can be seen in Table 1. During the training of our both agents catching and throwing, we linearly decay the exploration rate from Pe_{max} to Pe_{min} over 8000 episodes of training. The size of hidden layers is the same for both actor and critic and for both agents. For this experiment, we do not take in consideration collision which is part of a more complicated problem as we will explain in the later part of this paper.

Table 1: Parameters

Simulation time step	10ms
Replay Buffer size	1e5
Hidden Layer size (1)	512
Hidden Layer size (2)	256
Learning rate (Actor)	1e-4
Learning rate (Critic)	1e-3
Discount factor (γ)	0.95
Pe_{max}	1.0
Pe_{min}	0.20
P_{hand}	0.5s
D_{ratio}	70%
k_p	1200.0
k_v	70.0
w_1	0.12
w_2	1.0
w_3	0.29
w_4	9.0
w_5	1.4
Gravity (N/Kg)	9.81

4.1.1 States

The state of the agent contains information about the side of the body we will use and the ball concerned by the task. We get for the body state, the position p_{hand} and velocity v_{hand} of the hand, and a set of four joint angles $\theta_{\mathcal{J}}$ in radians with $\mathcal{J}$ = {neck,shoulder,elbow,hand}. Then, we get the position p_{ball} and the velocity v_{ball} of the ball. For the throwing agent we augment the state with the time left before throwing the ball t_{throw} and the velocity of the throw wanted $\bar{v}_{throw}$. As a result we have the catching state and the throwing state :

$s_{catching} = \{p_{hand}, v_{hand}, p_{ball}, v_{ball}, \theta_{\mathcal{J}}\}$

$$s_{throwing} = \{p_{hand}, v_{hand}, p_{ball}, v_{ball}, \theta_{\mathcal{J}}, t_{throw}, \bar{v}_{throw}\}.$$

4.1.2 Actions

For both policies, catching and throwing, an action represents a set of target joint angles $\bar{\theta}_{\mathcal{J}}$. We have for both agents an action $a = \{\bar{\theta}_{neck}, \bar{\theta}_{shoulder}, \bar{\theta}_{elbow}, \bar{\theta}_{hand}\}$. These target joint angles are used by the PD-controller to compute the torques to apply to the joints as shown on Figure 6.

$$\tau_i = k_p^i(\bar{\theta}_i - \theta_i) + k_v^i(\bar{\omega}_i - \omega_i) \tag{4.1}$$

where joint $i \in \mathcal{J}$, τ_i is the torque applied to it, $\bar{\theta}_i$ and θ_i are its joint angle target and actual respectively, $\bar{\omega}_i$ and ω_i are the joint angle velocity target and actual respectively. Finally, k_p^i and k_v^i are manually-specified gains. In this experiment, we set the same gains k_p and k_v for all joints and $\bar{\omega}_i$ is set to zero for all joints in $\mathcal{J}$.

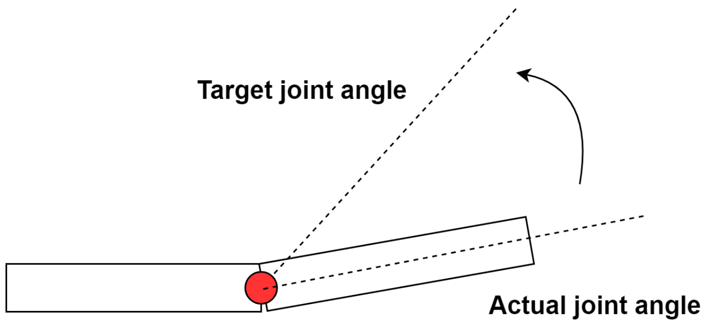


Figure 6: PD-Controller.

In our first implementation, we tried to use a target position for the hand as an action and to calculate the Inverse Kinematics to move toward it.

The problem of this method is that we can not get a real time simulation by querying an action at every simulation time step because inverse kinematics is very time consuming. That is the reason why we decided to directly use the target joint angles of our 2D half body to make our simulation step faster .

4.1.3 Environment of our Simulation

We used the early phase implementation of 2D juggling of the previous project [1] using Inverse Kinematics to move the hand of our character to the desired position and Finite Element Method (FEM) to calculate the actions needed to perform juggling. This implementation uses DART (Dynamic Animation and Robotics Toolkit) to create the physics-based simulation in C++. We modified this implementation and with Boost Library, used it in python where we created our own environment for our Juggling Simulation. Then, we define the boundaries of our environment and we render the scene using PyOpenGL as shown on Figure 7.

The biggest part of this work is in the modeling of sensors of the environment to detect all the different parameters. These one are used to calculate the rewards and to manage the catches and throws. Moreover, we need to have an understanding of all the juggling process to use these agents and give them the right parameters to make them succeed their own task.

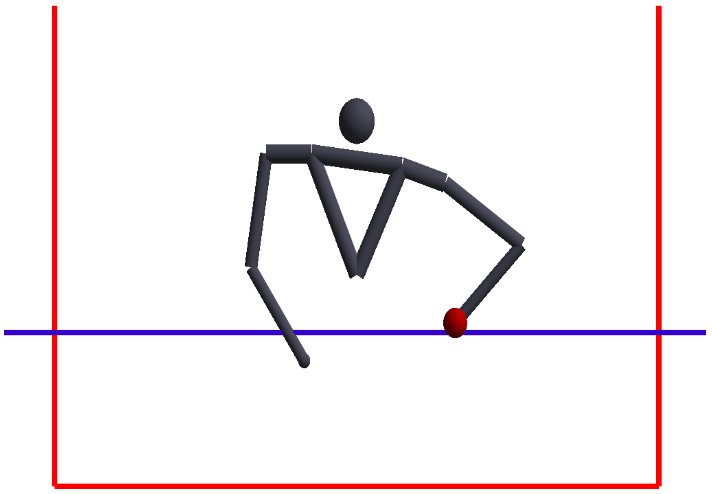


Figure 7: Environment of 2D juggling with the catching line (Blue) and the boundaries (Red).

4.2 Subtasks results

The training time needed for our agents to make their learning curve converge is around 1 hour 20 minutes for throwing and 2 hours for catching. The throwing task learning converges faster and is a far more stable than the catching task, but the final result for both is successful.

4.2.1 Throwing

We trained our throwing agent to move the hand with the grasped ball toward the the position wanted at the right velocity before the release time. This velocity is associated to the pattern we want to perform and we train it with Basic Patterns up to 7. Figure 9 shows the learning of the throwing task. At the start of the training, our hand fails to throw the ball at the right velocity resulting in a high negative reward. Finally our hand finds out a strategy which can balance value of negative rewards. At the final state of training, the reward average is around -2 with 40% of it due to the error in velocity, 25% due to error of position of throw wanted and 35% due to the effort of joints. We manually selected the reward weights to have the best and smoother agent as possible. Considering the velocity and position reward weights fixed, we found the effort reward weight being very sensitive to set as it can result in a very jerky result or a fail in being able to throw. A vertical velocity error directly impacts the time the ball will stay in the air. If this difference of time is too high, the ball may land in the wrong hand period. As we can have only one ball in the hand per period, the ball will not be caught and will fall on the ground. A horizontal velocity error changes the landing position of the ball. As long as this position is in the range of action of the hand receiving, the catch is possible, but if the ball lands outside of it then the catch will fail. Our agent can successfully throw a ball with a pattern up to 6 with a non critical error in velocity, and a limitation appears when trying to throw Basic Pattern 7 where he can not develop enough power to make the ball fly high enough.

4.2.2 Catching

We trained our catching agent to place his hand on a ball to catch it near the height wanted. The ball can come from any position at any velocity. As a result, the hand waits the ball to be in its range of action and rapidly moves to its position to catch it. The reward when the catch fails is inferior or equal to -1.0 and superior if the catch is a success. The closer the catch is to the position wanted, the higher is the reward. The hand successfully catches the ball as long as the ball is in its arm range and as close as possible from the height wanted. We manually selected the reward weights to ensure the catch while being smooth.

Catching is a very time sensitive task that makes the learning very unstable as we can see in Figure 8. We think that the number of frames where the ball is in catching range can be very short and is reduced if the velocity of the ball incoming is high or if we raise the simulation time step. The hand has to be precisely in the area of grasping of the ball on these key frames. Also, the sparsity of the catching reward and its discrete nature when the catch fails makes our early training set full of failing scenarios. This makes the training more difficult and unstable as successful catches will not happen very often. But in the 2D simulation, the number of catches made in the early phase of training is still high enough to make our training succeeds. If we set a low reward effort weight, our catching agent tries to sweep the area very fast near the position of landing. This strategy gives a high probability to catch the ball and also makes the learning more stable but full of jerks and unnatural. In this work, we will keep the value of effort reward weight where our agent performs the smoother catches.

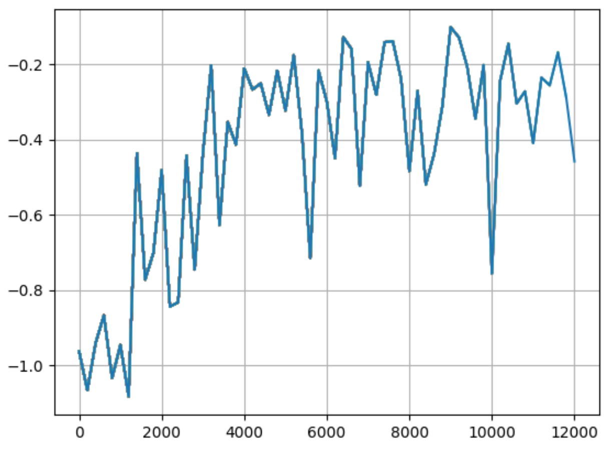


Figure 8: Learning curve for catching task in 2D simulation.

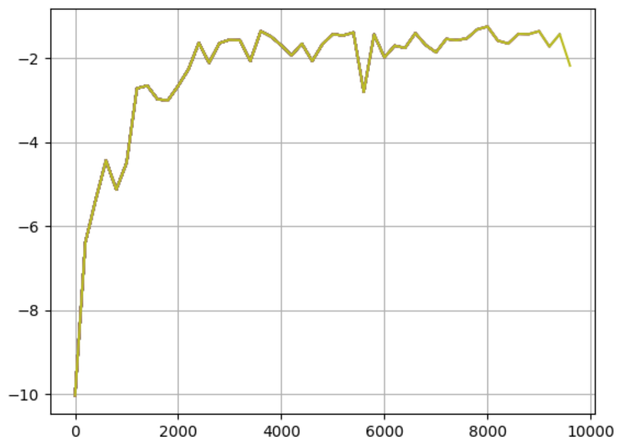
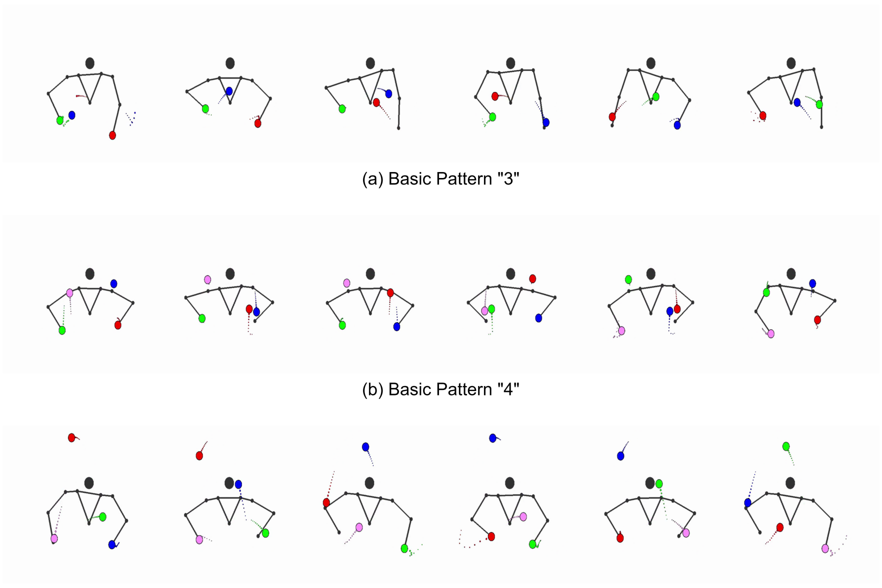


Figure 9: Learning curve for throwing task in 2D simulation.

4.3 Performing juggling

To demonstrate our result, we will test our juggling implementation by juggling the common Basic Patterns, then more complex Siteswap and finally by adding some new balls while juggling. Finally, we tested the robustness of our juggling by adding some wind on balls.


(a) Basic Pattern "3"

(b) Basic Pattern "4"

(c) Siteswap "53"

Figure 10: Juggling patterns.

4.3.1 Results

After these agents learned successfully to perform their respective subtasks, we apply them to both arms and we switch between both agents as explained in Figure 4. The catching agent succeeds in catching the ball thrown from the same or other hand. The throwing agent succeeds in tossing the ball with the right velocity and to make it land on the aimed hand at the time wanted for all patterns up to 7. Finally, the result of the combination of both agents appears to be quite smooth when juggling continuously.

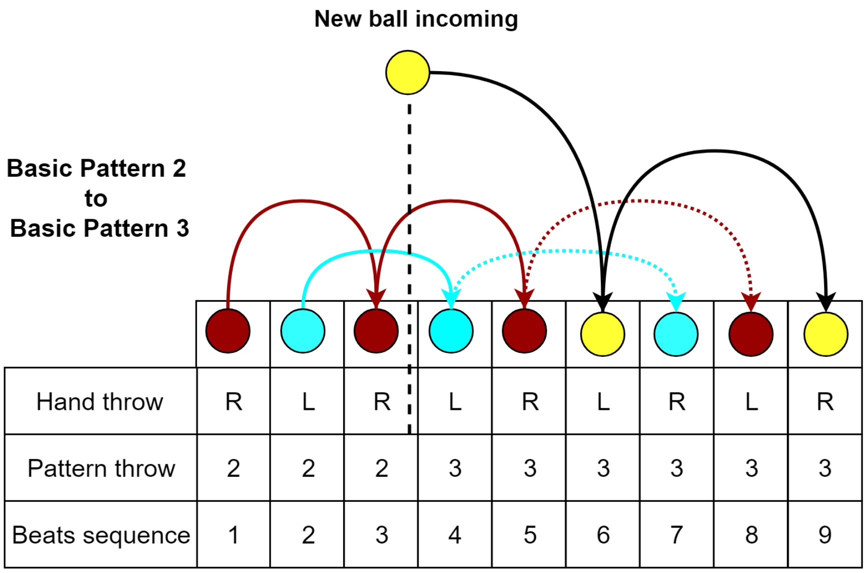
Figure 10 shows some results of our juggling simulation. The basic pattern 3 (a) consists in throwing a ball to the other hand at every beat time. Basic pattern 4 (b) is an even number, so the thrown will be for the same hand. Finally the Siteswap 53 (c) consists in throwing a ball from the left hand with a basic pattern 3 and from the right hand with a basic pattern 5.

When trained with different reward weights for throwing, our character can show some limitations with patterns superior or equal to 7, as it does not have enough power to throw it high enough. As a result, the hand period when the ball is in catching range, is already occupied by another ball and the juggling fails.

We test the robustness of our catching agent by adding some wind on balls while in the air, adding a constant horizontal acceleration and making them landing further or closer from our character. Our character succeeds to catch all balls as long as they land in his range of action and still have enough time to perform all throws corresponding to Basic Patterns up to 6.

4.3.2 Add new ball while juggling

How can a juggler receive a new ball while juggling. Every juggler would probably have a different answer to this scenario which will probably depends on the actual pattern performed, with how many balls, how much time before receiving this new ball and what is its position and velocity. In this first work, we adopted the common method which consists in increasing



Hand throw	R	L	R	L	R	L	R	L	R
Pattern throw	2	2	2	3	3	3	3	3	3
Beats sequence	1	2	3	4	5	6	7	8	9

Figure 11: Transition from Basic Pattern 2 to 3. The new yellow ball will arrive at beat number 6. We have to change all patterns performed before. The number of throws changed is equal to the previous number of balls juggled, so we have to change the throws at beat number 4 and 5. The new pattern performed is equal to the new number of balls juggled.

the pattern performed before receiving the new ball, see Figure 11. In juggling shows, the juggler knows that he will receive a new ball. So as shown on the figure, he frees the slot where he will receive the new ball which is in our case the yellow one. To do so, we can choose to change the pattern performed and set it to the new number of balls juggled. Finally, someone else throws the new ball in this empty slot. This method is probably the safest to perform this task. Other methods exists but they require to modify our period of throw as we will discuss about after.

Chapter 5

Toward a 3D juggling

After creating a 2D agent juggling, we want to perform a 3D simulation which could bring much more naturalness and possibilities of juggling movements. At first, we need to add 5 more degrees of freedom to our body joints in order to cover the space and to extend the position and velocity by one more axis. It means that the number of states and actions will be increased but also that our previous reward design may have to be modified for our both tasks.

5.1 Catching

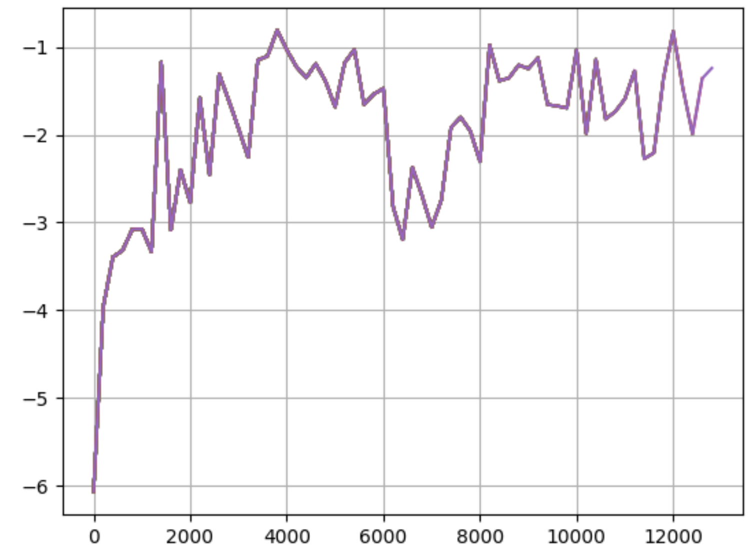
Our catching agent in 2D has an unstable learning and probably needs some optimization regarding the reward design and the training. That is why in 3D we have to modify our approach.

As said in section 4.2.2, the early training set will be mainly composed of fails scenarios and the transition to 3D makes this problem even more important. Cases were the ball is caught randomly almost never happen and as a result, our training set is composed of fails all the long of the training and our agent can not learn. To fix this problem, we add a new reward to encourage the hand to stay at the intersection of the trajectory of the ball and the catching line. We could think that this ensures the catch as the hand is supposed to stay on trajectory of the ball, but surprisingly, our hand failed to catch it very often, probably due to the jerky movements. That is why we keep the catching reward to make our agent ensure the catch. We believe that this reward will encourage the agent to stay on the trajectory of the ball and when the ball will be very close to our hand, the catching reward will permit to the agent to do any move necessary to catch the ball.

By adding this new reward, we get a training much more stable than in the 2D version designed with the discrete and sparse reward for catching the ball, as seen on Figure 12. But the result of the training is also very sensitive to parameter changes and some of our training sessions resulted in an agent missing the ball quite often. We will discuss some ways to fix this problem later on.

5.2 Throwing

The training of our throwing agent was very stable in our 2D simulation, but the transition to 3D had a negative impact on it and on its performance, as seen on Figure 12. For now our throwing agent can throw some patterns up to 5, but can not find a balance between power and accuracy of the throw for higher patterns. We still need to investigate the reasons and try different parameter values and rewards to fix this problem. We believe this problem could also be solved by trying another Reinforcement Learning algorithm.



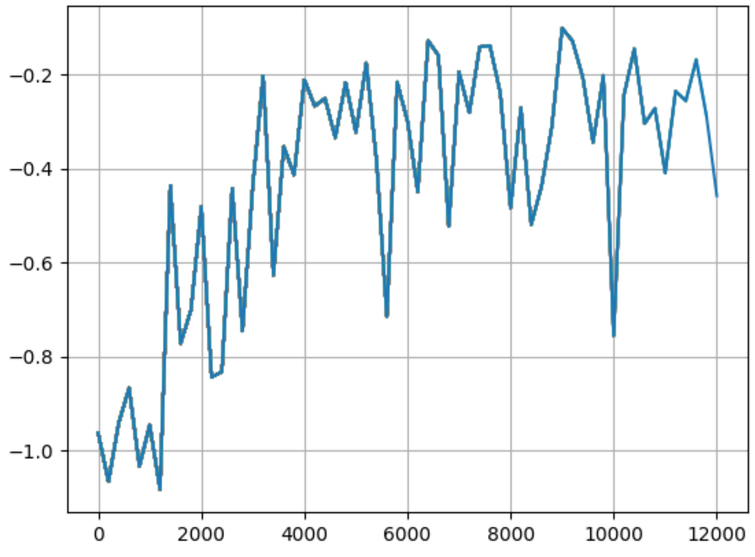


Figure 12: Learning curves for catching task (top) and throwing task (bottom).

5.3 Results

A result of our 3D juggling simulation can be seen on Figure 13 with a pattern ’5’ performed. Our both agents both succeed in their respective task. The throwing agent loses in power or accuracy compared to the 2D version and can throw some Basic patterns up to 5. We then recreate the process of juggling exactly as done before by applying them sequentially on both sides of the body. Our character can successfully juggle patterns from 2 to 5 and can still perform when applying wind on balls in the air.

The training of the catching agents give some results very different for the same parameters. In our latest tests after 10 seconds of juggling, our catching agent was sometime missing the catch resulting in a fail. Regarding the throwing agent, it has a big margin of errors which most of the time can be caught back by the catching agent for Basic patterns up to 5. But For higher patterns, the ball is thrown sometime outside of the boundaries resulting in a fail.

As a result, we still need to work on improving performances for both tasks with a priority for the catching agent which is critical for juggling continuously. After fixing these problems, we could be able to perform more complex Siteswaps and different kind of throws and catches. This opens many rooms of exploration for all the possibilities offered by the world of Juggling.

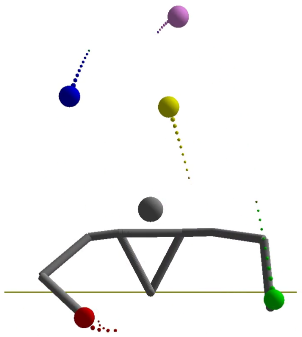
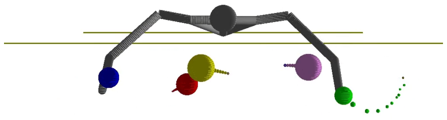
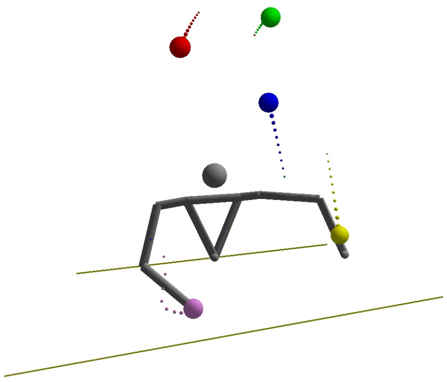




Figure 13: Basic Pattern (5) in 3D from a front, top and right view respectively.

Chapter 6

Discussion and Conclusion

We presented a method to learn a catching and throwing task in a physics-based simulation using Deep Deterministic Policy Gradients. Finally, we recreated the all process of juggling by using our agents sequentially, respecting the mathematics of juggling. In our simulation, Reinforcement Learning allows us to have a real time Juggling simulation able to adapt to some external forces on balls and to perform some patterns continuously with success. Then, we presented an improvement of our simulation going from 2D to 3D by discussing about the modifications needed for the rewards.

The unstable learning of the catching agent by using a discrete and sparse reward has an impact on his ability to catch some balls in difficult situations. We found a solution in our 3D simulation to make this training more stable, but this still needs to be improved to ensure every catches. Concerning the throwing agent, our character fails to throw patterns higher than 7 or 5 for the 2D and 3D version respectively which requires more power. An easy solution could be to raise the gain of the PD-controller k_p and k_v to make the joints moving faster but this would lead to more unnaturalness. There is also a lot of improvements to do in order to fix the still jerky movements of our character, as setting customized gain for each joint and trying another reward effort function. We plan to use another Reinforcement Learning algorithm named Proximal Policy Optimization (PPO) that has shown many good results in physics-based simulation providing a smoother and always increasing learning curve compared to Deep Deterministic Policy Gradients.

Another room of exploration could be to consider collisions and to add some balls while juggling and without warning our character of the change. An unprepared change in pattern appears in both of these cases where a free beat slot will not be available and our juggler should be able to adapt to keep on juggling. In order to do this, it would require to plan the timing, power and direction of our future throws. We could use a High and Low level controller as demonstrated by Peng et al. [13] applied to locomotion planning. In this implementation, the High level controller would plan the timing and velocities of throws to perform, and give it to the low level controller which is the throwing agent.

Finally, our agents are trained to catch at a fixed height and to throw at a fixed position. It could be interesting in a future work to train him to throw and catch at any position allowing more complex juggling patterns.

Juggling is a skill which offers a lot of possibilities for physics-based simulations and this work covers only the basics of it. In the end, solving and simulating all these juggling possibilities would help us to have a better understanding in human simulation and also to help jugglers to find new way of improving this art.

References

[1] S. Lee, R. Yu, J. Park, M. Aanjaneya, E. Sifakis, and J. Lee, "Dexterous manipulation and control with volumetric muscles," *ACM Transactions on Graphics (SIGGRAPH 2018)*, vol. 37, no. 57, 2018.

[2] P. J. Beek and A. Lewbel, "The science of juggling - studying the ability to toss and catch balls and rings provides insight into human coordination, robotics and mathematics," *Scientific American*, vol. 273, no. 5, 1995.

[3] J. Kalvan, "The human limits - how many objects can be juggled," 2018.

[4] C. E. Shannon, "Scientific aspects of juggling," in *Claude Elwood Shannon: Collected papers* (A. D. Wyner and N. Sloane, eds.), New York: IEEE Press, 1993.

[5] P. Burget and P. Mezera, "A visual-feedback juggler with servo drives," *The 11th IEEE International Workshop on Advanced Motion Control*, 2010.

[6] J. Kober, M. Glisson, and M. Mistry, "Playing catch and juggling with a humanoid robot," *12th IEEE-RAS International Conference on Humanoid Robots*, 2012.

[7] S. Jain and C. K. Liu, "Interactive synthesis of human-object interaction," in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, (New York, NY, USA), pp. 47–53, ACM, 2009.

[8] K. Ding, L. Liu, M. van de Panne, and K. Yin, "Learning reduced-order feedback policies for motion skills," in *Proc. ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2015.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, “Playing atari with deep reinforcement learning,” *CoRR*, vol. abs/1312.5602, 2013.

[10] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver, “Emergence of locomotion behaviours in rich environments,” *CoRR*, vol. abs/1707.02286, 2017.

[11] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, “Deepmimic: Example-guided deep reinforcement learning of physics-based character skills,” *ACM Transactions on Graphics (Proc. SIGGRAPH 2018 - to appear)*, vol. 37, no. 4, 2018.

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *CoRR*, vol. abs/1509.02971, 2015.

[13] X. B. Peng, G. Berseth, K. Yin, and M. van de Panne, “Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning,” *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)*, vol. 36, no. 4, 2017.

Acknowledgements

I would like to thank all the people who helped me to write this Master Thesis. At first, I am very grateful to my Professor Jehee Lee for giving me the opportunity to be in his lab, to study the field I wanted to and to discover far more than I expected, from the Korean life with his students to the world of research. This experience really broadened my eyes on careers and fields I would have never known about and it helped me to find my way.

I am also so thankful to all my colleagues who helped me so much during these two years with them and I litterally would have not survive without them. Thanks again to Jungdam Won, Hwangpil Park, Ri Yu, Sehee Min, Seunghwan Lee, Minseok Kim, Jaedong Lee, Soohwan Park, Jungnam Park, Seyoung Lee, Hoseok Ryu, Jongho Park, Kwanyu Kim, Yongsung Kim, Sunmin Lee. Everything was Awesome !

Finally, I thank all my family. This has probably been a long three years far from you abroad but thank you for supporting me and all my choices. The last thought is for my grandfather who I hope is proud of me and my thesis from where he is.