PHYSICAL HAND ANIMATION WITH MACHINE LEARNING

MAKİNE ÖĞRENMESİ İLE FİZİKSEL EL ANİMASYONU

Tarık CANTÜRK

Assist. Prof. Dr. ZÜMRA KAVAFOĞLU
Supervisor

Submitted to Institute of Informatics of Hacettepe University
as a Partial Fulfillment to the Requirements
for the Award of the Degree of Master of Science
in Computer Graphics

2020

ABSTRACT

PHYSICAL HAND ANIMATION WITH MACHINE LEARNING

Tarık Cantürk

Master of Science, Department of Computer Graphics

Supervisor: Asst. Prof. Dr. Zümra Kavafoğlu

August 2020, 77 pages

Hands are the essential limbs of humans which they use for interacting with their environments. Catching, holding, moving, touching, and many other interactions are done by hand. The hand has a highly complex anatomical structure. It is a quite complicated task to model the hand movements, considering the bones of the fingers, joints, muscles, and tendons that connect them to each other and move them.

Several motion capture systems are used to transfer hand motions to the digital environment. However, it's harder to capture a catching motion with these systems, compared to capturing hand interactions with steady objects. Besides, only kinematic animations can be generated with motion capture systems, and these kinematic animations may not be usable for different catching scenarios. Therefore, employing physics-based animation techniques for generating hand motions is needed.

To generate hand motions with physics-based animation techniques, the physical model of the hand must be created. We present a physical hand model with muscles and soft tissues on a skeletal structure. The presented model is intended to create realistic physical interactions and also to be efficient enough.

Recently, great accomplishments have been achieved in computer animation field with the employment of machine learning techniques. We present a framework that generates catching motions for the proposed physical hand model, by using deep reinforcement learning techniques.

To catch a thrown object, multiple body parts are required to work in coordination. While our main focus is to generate proper physics-based hand motions, we also work on synthesizing arm motions that are essential for taking the hand to the correct interception point with the right orientation.

It's been addressed in the literature [1, 2] that catching motion can be divided into smaller phases. In this way, we handle the catching motion in two phases and developed a controller brain for each phase by using deep reinforcement learning. One of these brains is designed to move the arm for getting prepared for the catching motion. And then the other one is designed to control the hand for accomplishing the actual catching movement. In addition to these, a third brain is generated with deep reinforcement learning, that manages the working time of these two brains.

The results of the proposed framework is evaluated and compared with other configurations by several experiments. Moreover, user test studies have been conducted for evaluating the naturalness of the resulting motions.

Keywords: Physics-based animation, hand animation, machine learning, reinforcement learning, curriculum learning, physics-based hand model.

ÖZET

MAKİNE ÖĞRENMESİ İLE FİZİKSEL EL ANİMASYONU

Tarık Cantürk
Yüksek Lisans, Bilgisayar Grafiği Bölümü
Tez Danışmanı: Dr. Öğr. Üyesi Zümra Kavafoğlu
Ağustos 2020, 77 sayfa

Eller, insanların çevreleriyle etkileşim için kullandıkları en önemli uzuvlarıdır. Bir nesneyi yakalamak, tutmak, taşımak, ona dokunmak ve daha birçok etkileşim el ile yapılmaktadır. El, anatomik olarak çok karmaşık bir yapıdadır. Her bir parmaktaki kemikler, eklemler, onları birbirine bağlayan ve hareketi sağlayan kas ve tendonlar hesaba katıldığında, bir işlevin gerçekleştirilmesini modellemek için yapılması gereken işlemlerin karmaşıklığı ortaya çıkmaktadır.

El hareketlerini bilgisayar ortamına aktarabilmek için çeşitli hareket yakalama sistemleri kullanılmaktadır. Ancak fırlatılan bir nesnenin yakalanması hareketinin bu sistemler ile üretilmesi işlemi durağan nesnelerle yapılan hareket yakalama işlemlerine göre çok daha zordur. Ek olarak, başarılı bir hareket yakalama verisi ile ancak kinematik animasyon oluşturulabilir ve bu kinematik animasyonlar farklı yakalama senaryoları için kullanılmaya uygun olmayabilir. Bu nedenle, el hareketlerinin fizik tabanlı animasyon ile sentezlenmesi ihtiyacı doğmaktadır.

El hareketinin fizik temelli animasyonunu üretebilmek için elin fiziksel modelinin oluşturulması gerekir. Bu tez çalışmasında, bir iskelet yapısı üzerinde kas ve yumuşak dokular bulunan fiziksel bir el modeli sunulmaktadır. Sunulan modelin hem gerçekçi fiziksel etkileşimler oluşturması hem de işlem hızı bakımından verimli olması amaçlanmıştır.

Günümüzde, makine öğrenmesi yöntemlerinden faydalanılarak, bilgisayar animasyonu alanında çok başarılı çalışmalar yapılmaktadır. Biz de bu tez çalışmasında, sunduğumuz el modelinin bir nesneyi yakalaması için gerekli hareketleri, derin pekiştirmeli öğrenme yöntemleriyle üreten bir sistem ortaya koymaktayız.

Fırlatılan bir nesnenin yakalanması hareketi, bir çok vücut parçasının koordineli olarak çalışmasıyla elde edilir. Bizim bu çalışmada esas odağımız uygun fizik tabanlı el hareketlerinin üretilmesi olmakla birlikte, eli nesneyle kesişeceği doğru konuma ve oryantasyona getirmek için gerekli olan kol hareketlerinin sentezlenmesi de çalışmaya dahil edilmiştir.

Yakalama hareketinin küçük aşamalara bölünerek ele alınabileceği literatürde gösterilmiştir [1, 2]. Biz de bu çalışmada yakalama hareketini iki aşamada ele alıp, her bir aşama için derin pekiştirmeli öğrenme ile farklı bir kontrolcü beyin geliştirdik. Bu beyinlerden ilki, kolu yakalama hareketine hazırlanacak biçimde hareket ettirmek için tasarlanmıştır. İkinci beyin ise, eli kontrol ederek esas yakalama işlemini gerçekleştirmek için tasarlanmıştır. Buna ek olarak, bu iki kontrolcünün çalışma zamanlarını yönetecek üçüncü bir beyin de yine derin pekiştirmeli öğrenme ile üretilmiştir.

Çalışmamızda ortaya koyduğumuz sistemin sonuçları çeşitli deneyler aracılığıyla değerlendirilmiş ve farklı konfigürasyonlarla karşılaştırılmıştır. Buna ek olarak üretilen yakalama hareketinin gerçekçiliğini değerlendirmek için kullanıcı testleri gerçekleştirilmiştir.

Anahtar kelimeler: Fizik-temelli animasyon, el animasyonu, makine öğrenmesi, pekiştirmeli öğrenme, aşamalı öğrenme, fiziksel el modeli.

ACKNOWLEDGEMENT

Foremost, I would like to express my sincere gratitude to my advisor Assist. Prof. Dr. Zümra Kavafoğlu for her continuous support during my master study and research, for her patience, motivation, enthusiasm, and immense knowledge. Her guidance helped me all the time through the research and writing of this thesis. I could not have imagined having a better advisor and mentor for my master study.

Besides my advisor, I would also like to thank the rest of my lecturers in my M.Sc. education: Assist. Prof. Dr. Serdar Arıtan, Prof. Dr. Haşmet Gürçay, Assist. Prof. Dr. Ufuk Çelikcan. Also I would like to thank Assoc. Prof. Dr. Uğur Güdükbay for their valuable comments on the thesis.

I have been working at Turkish Aerospace Company for 8 years, my sincere thanks also go to them. They supported my graduate education and gave me the chances to use my experience to develop better products. I thank my colleagues and manager in Turkish Aerospace: Güray Yıldız for his support all time, Erkan Çağır, Serkan Gencer, İbrahim Ardıç and Dr. Emre Akyılmaz for their ultimate support and for all the fun we have had in the last six years.

Words fall short to thank my parents Fatma and Mehmet Cantürk for their efforts on my education. I feel fortunate to be the child of such supporter people. I also thank my dear brother Deniz Cantürk, my mother-in-law and father-in-law and all other members of my family for their great support. And thanks a lot to all my friends for cheering me up in this long period.

The last thanks are to the most precious, to my lovely wife Yeliz, with whom our hearts beat together. Thanks to my sweetheart for always supporting me, showing me different perspectives and making my life happier and more peaceful. And our little "angel" ...

TABLE OF CONTENTS

Page

ABSTRACT i

ÖZET iii

ACKNOWLEDGEMENT v

TABLE OF CONTENTS vi

LIST OF TABLES viii

LIST OF FIGURES ix

1 INTRODUCTION 1

1.1 Motivation and Scope of the Work 1

1.2 Thesis Outline 3

2 BACKGROUND 4

2.1 Reinforcement Learning 4

2.1.1 Basic Information About Reinforcement Learning 4

2.1.2 Proximal Policy Optimization 5

2.1.3 Curriculum Learning 7

2.1.4 Hierarchical RL systems 8

2.1.5 Unity ML Agents 10

2.2 Physics-based Animation 14

2.2.1 General Physics-based Animation 14

2.2.2 PD Controller 16

3 RELATED WORK 18

4 HAND AND ARM MODEL 22

4.1 Hand Model 22

4.2 Arm Model 28

5 LEARNING TO CATCH WITH HIERARCHICAL REINFORCEMENT LEARNING 29

5.1 System Overview 29

5.2 Reactive Learner with Proximal Policy Optimization 31

5.3 Proactive Learner with Curriculum Learning 34

5.4 Timing Learner with Hierarchical Deep Reinforcement Learning 40

6 IMPLEMENTATION AND RESULTS 44
6.1 Implementation Details 44
6.2 Training Parameters 49
6.3 Proactive Brain Comparison: Train versus Heuristic with Skeleton Hand Model 52
6.4 Proactive Brain Comparison: Trained versus Heuristic with Soft-body Hand Model 53
6.5 Hand Model Comparison: Skeleton versus Soft Body 55
6.6 Meta Brain Comparison: Trained versus Heuristic 57
6.7 Reactive Brain Reward Comparison 60
6.8 User Test Results 63
7 CONCLUSION 67
7.1 Limitations 68
7.2 Future Work 69
REFERENCES 70
CURRICULUM VITAE 77

LIST OF TABLES

4.1 Hand model properties. Finger joints do not have rotational degrees of freedom about the y axis [3]. 24
4.2 Arm model properties [4]. 28
5.1 The absolute value of processed action values are added to some DOFs, where the movement is restricted to only positive direction. 38
6.1 Training parameters 50
6.2 User test results for different meta brains. 64
6.3 User test results for the proactive phase. 66

LIST OF FIGURES

2.1 RL training cycle [5]. 4
2.2 Example of a curriculum for mathematic [6]. 7
2.3 Demonstrate a curriculum training scenario where gradually a longer wall blocks the path to the target [6]. 8
2.4 Cumulative reward of RL training session [6]. 8
2.5 Simplified block diagram of ML-Agents. [7] 12
2.6 Simplified block diagram of ML-Agents [7]. 13
2.7 PD Controller Scheme 16
2.8 *Kp/Kd* sample chart [8]. 17
4.1 Hand anatomy: (a) skeleton, (b) skeleton with muscles and tendons, (c) skeleton with muscles, tendons, and fat [9]. 22
4.2 Hand skeleton model with joint degrees of freedom. 23
4.3 Hand skeleton from front (a) and side (b) 25
4.4 Hand with soft-bodies (a) and visual mesh (b) 26
4.5 Muscle structure of (a) a lumbricale muscle and (b) intrinsic muscles of the thumb. 27
4.6 Deformation model for object-muscle interaction. 28
4.7 Arm model. Numbers indicate the joints in Table 4.2. 29
5.1 The system overview diagram. 31
5.2 Hand skeleton and thrown objects. 35
5.3 Curriculum learning timeline of the used object shape. 36
5.4 Sensor positions on the hand. There are 28 inside sensors (green) and 7 outside sensors (red). 37
5.5 The relation of timing learner with reactive and proactive brains. 41
6.1 Proactive learner environment is shown from different angles. 45
6.2 Some key steps from the Reactive Learner environment. 47
6.3 Some instants from the Timing Learner environment. 48
6.4 Tensorboard chart of the Reactive Learner training process. 51
6.5 Tensorboard chart of the Proactive Learner training process. 51
6.6 Tensorboard chart of the Timing Learner training process. 51
6.7 The comparison of trained proactive brain (up) and heuristic proactive brain (down) for catching a ball with skeleton hand model. 52

6.8 The comparison of trained proactive brain (up) and heuristic proactive brain (down) for catching a rectangular prism with skeleton hand model. 53
6.9 The comparison of trained proactive brain (up) and heuristic proactive brain (down) for catching a ball with the soft-body hand model. 54
6.10 The comparison of trained proactive brain (up) and heuristic proactive brain (down) for catching a capsule with the soft-body hand model. 54
6.11 The comparison of the trained proactive brain for catching a ball with the soft-body hand model (up) and the skeleton hand model (down). 56
6.12 The comparison of the trained proactive brain for catching a capsule with the soft-body hand model (up) and the skeleton hand model (down). 56
6.13 The comparison of the training charts of the proactive brain with the soft-body hand model (blue) and the skeleton hand model (cyan). 56
6.14 The comparison of the trained meta brain and the heuristic meta brain for catching a sphere with the soft-body hand model. 58
6.15 The comparison of the trained meta brain and the heuristic meta brain for catching a capsule with the soft-body hand model. 59
6.16 The effect of the higher weight on the palm forward reward. 61
6.17 The effect of calculating the displacement penalty according to hand-object distance. 62
6.18 User test results for different meta brains. 64
6.19 User test results for proactive phase 66

1 INTRODUCTION

1.1 Motivation and Scope of the Work

From the very beginning of their lives, humans learn to use their hands gradually in order to interact with their environment. Hands provide tremendous interaction capabilities for achieving a wide variety of tasks, including fine motor skills like grasping and manipulating or gross motor skills like throwing and catching an object. Imitating the motion of this major apparatus of humans has long been a popular interest for computer animation researchers. As the virtual environments take more part in our lives, the requirement of efficient and high quality hand animations increased. Real time applications have an unpredictable nature, for which it is hard to model the interactions with the environment beforehand. Physics-based motion generation techniques offer a general solution for this problem.

Catching is a complicated motion, accomplished with the coordinated movement of different body parts. Humans achieve a successful catching as follows: First they observe the approaching object and estimate an interception point. If they are far from this interception point, they move towards and adjust their hand there. As the ball gets closer to the anticipated point, they fine tune the position and orientation of the hand. When the ball contacts the hand they grasp it as quick as possible to avoid its bouncing back and falling. Each submovement in this process constitutes a different motion synthesis problem. Different from other hand-object interaction tasks, object of interest is non-steady in catching, therefore it is also of great importance to adjust the timing of these submovements.

We propose a framework for generating arm and hand motions for catching objects with several geometries in real time. The main focus of our work is especially on finger motions for a stable grasping of the object at the instant of hand-object contact. For handling the real time interactions with the object, hand is controlled physically, while the arm is animated kinematically for efficiency concerns. We examined the catching motion in two distinct phases, as *reactive* and *proactive*, each of which is modeled as a different learning problem. The reactive phase includes the preparatory motions of the kinematic arm while the proactive phase includes the actual catching behavior with the physically modeled hand. For learning the arm motions of the reactive phase, we use deep reinforcement learning with proximal policy optimization and we use curriculum learning for learning to control the finger motions for catching objects of different geometry. For coordinating the learned motions for these phases, we devised a third learning system. Instead of devising only one learning system for whole catching behavior, we preferred a partitioned system for several reasons. We obtain an improvement in the performance by reducing the search space complexity. Moreover, we have the freedom of designing each subproblem to meet its specific requirements with a different learning algorithm.

For generating a realistic and stable grasping motion at the catching moment, we modeled a two layer hand model that consists of a rigid body chain covered with a deformable model for imitating the skeleton and the soft tissue of the human hand. The deformable model is a combination of triangular meshes of point masses tied to the rigid bodies of the skeleton model. We achieve the deformation by the mass-spring simulation of point masses with Proportional Derivative Controller. We model the arm as a simple kinematic chain for the sake of efficiency.

The key contributions of our work are as follows:

- A novel framework for generating a catching motion including finger motions for stable gripping. To the best of our knowledge, this is the only work in the computer animation literature that includes generating finger motions for catching behavior.
- A detailed deformable model for the hand including palm for achieving a more stable hand-object interaction. Although, there are more detailed deformable hand models proposed in the literature, they are not designed for hand-object interaction problems.
- Handling the catching motion generation problem by devising three different learning environments for different parts of the problem.
- A novel touch sensor based reward mechanism for learning natural finger motions.
- Using hierarchical reinforcement learning for handling the coordination of different phases of catching motion.

1.2 Thesis Outline

The remainder of the thesis is organized as follows.

Chapter 2 first provides background on machine learning and reinforcement learning, together with the essential components of reinforcement learning. Then it provides background on physics-based animation.

Chapter 3 presents related work in the computer animation literature about physics-based hand object interactions.

Chapter 4 describes the hand and arm models used in our work.

Chapter 5 presents the main learning framework, which is divided into three core components. Each component's ML model is explained in detail, together with the relationships between them.

Chapter 6 provides implementation details, specifically the parameters of the training environments and their effects on the produced results. Besides, this chapter discusses the user tests, their results, and compares different approaches for modeling and implementation.

Chapter 7 concludes the thesis with possible further research directions.

2 BACKGROUND

2.1 Reinforcement Learning

2.1.1 Basic Information About Reinforcement Learning

Machine learning (ML) is the designated model of a system to make predictions from input by processing data with mathematical and statistical methods.

Nowadays, there is a lot of machine learning algorithms and methodologies. According to the learning method, three groups can be mentioned; *supervised*, *unsupervised*, and *reinforcement* [10].

The basic working mechanism of Reinforcement Learning (RL) is as follows: An agent performs an action according to its observations; it is called *policy*. After this action, a reward value is given to the agent. According to the reward value, the agent is trained and learns the outcome of its decision. The agent tries to improve its policy gradually to choose the best options according to the rewards. In a stochastic task, each action must be attempted many times to get a reliable estimate of its expected earnings. For example, a newborn baby finds the truth through trial and error. When a little boy touches the stove, he understands that it is a bad thing and knows that he will not go there again. RL agents act purposefully, they all have clear goals. They can feel the characteristics of their environment and choose the actions that will be effective in their environment.

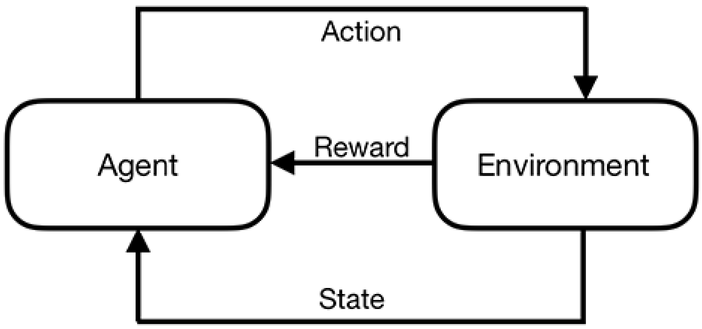


Figure 2.1: RL training cycle [5].

The peculiarity of this system is that it conditions artificial intelligence to a result. This result is always the highest cumulative reward. It is seen as a punishment for artificial intelligence, as it does not get less, a different reward or no reward at all. Therefore, it can be said that motivation always works in one direction. One of the most exciting aspects of RL is its solid and efficient interactions with other engineering and science disciplines. RL has a decades-long trend in artificial intelligence and machine learning for further integration with statistics and other mathematical topics.

Process steps for RL

- Since the agent has insufficient knowledge about the environment in which it is located, it makes observations to establish a link between cause and effect.
- After the observation is over, the agent is forced to choose between options. Since the highest reward is intended, it takes action by making a decision that will be most suitable for that reward.
- After the first step is over, the agent takes an action for that observation and looks at new options. The important part is that the artificial intelligence has learned how to make the right decision to reach its goal after the first step.

2.1.2 Proximal Policy Optimization

Policy gradient methods are crucial for deep neural networks for solving the problem. It is used in many different areas, from video games to robotic control. Because of the need to tune step size, it is not easy to get good results with policy gradient methods. If the step size is too small, progress will be slow. If large step size is used, the output signal will be too noisy. Besides, sample efficiency gets too low, and learning simple tasks takes too much time.

Gradient descent can be applied in supervised learning successfully with a small tuning of hyperparameters. However, significant effort is required to tune hyperparameters in RL [11]. The cost function is easily implemented in supervised learning and made a gradient descent on it. Therefore, with a small tuning of hyperparameters, an excellent result can be achieved. In reinforcement learning, the success path is complicated; it is hard to debug the algorithm.

Proximal Policy Optimization (PPO) [12] also calculates an update to minimize cost function at every step. During this calculation, it tries to keep the deviation relatively small from the previous step. PPO is establishing a balance among simplifing application, adjustment, and sample complexity [11].

Natural policy gradient is commonly used to solve the convergence problem. However, the natural policy gradient has a second-order derivative matrix, which is not scalable for the large-scale problem. Its complexity is too high for everyday tasks. PPO formalizes restrictions as a punishment in the objective function, instead of applying complex restrictions. The first-order optimizer can be used to optimize the target as a gradient descent method. Therefore, the restrictions are avoided. While avoiding complex restrictions, the calculation is much more comfortable and faster.

PPO uses a novel objective function that is not similar to other algorithms [11]:

$$L^{CLIP}(\theta) = E_t[\min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)],$$

- θ denotes the policy parameter,
- E_t denotes the empirical expectation over timesteps,
- r_t denotes the probability ratio,
- A_t is an estimator of the adventage function at time t,
- ϵ denotes a hyperparameter, say 0.2.

This objective implements updating stochastic gradient landing with a compatible trust region. It simplifies the algorithm by removing the Kullback–Leibler (KL) penalty and updating adaptively. According to the tests, this algorithm performs better results in continuous control tasks, and it is much easier to implement [11].

2.1.3 Curriculum Learning

Curriculum learning is one of the learning algorithms that, learning task starts with an easy one, and task difficulty increases along training time [13]. Humans learning style is very similar to this algorithm. However, we can not transfer it to a neural network. Therefore, we need to train all data for all difficulties from the beginning. For example, our educational system and their rankings are classified. We firstly learn simple arithmetic operations; after that, we learn algebra. After learning algebra, we learn calculus, and this learning style goes on [6]. Our previous experience helps us to learn new tasks more efficiently according to trying to learn it from scratch (see Figure 2.2). In machine learning algorithms, a similar principle works. To learn a complex task, tasks are ordered from easy to difficult to reduce training time and provide more successful results than unordered tasks.

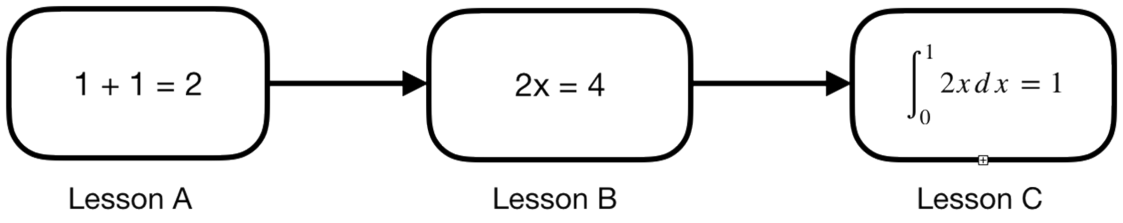


Figure 2.2: Example of a curriculum for mathematic [6].

Curriculum learning technique and its benefits are explained as follows in [6]. When thinking about how RL works, the primary learning signal is a scalar reward that is received from time to time throughout the training. This reward can often be sparse and rarely achieved in more complex or challenging tasks. For example, consider a task that an agent must push a block in place to scale a wall and achieve a goal (see Figure 2.3). When training an agent to perform this task, the starting point will be a random policy. This initial policy will likely involve the agent working in circles. It will never or rarely scale the wall properly to receive the reward. Starting with a more straightforward task, such as moving towards an unobstructed goal, the agent can quickly learn to accomplish the task. From there, the training environment can slowly add task difficulty by increasing the wall size, until the agent can complete the almost impossible task of scaling the wall initially.

To see curriculum learning in action, observe the two learning curves below (Figure 2.4). Each curve indicates the reward by time for a brain trained using PPO, with the same train-

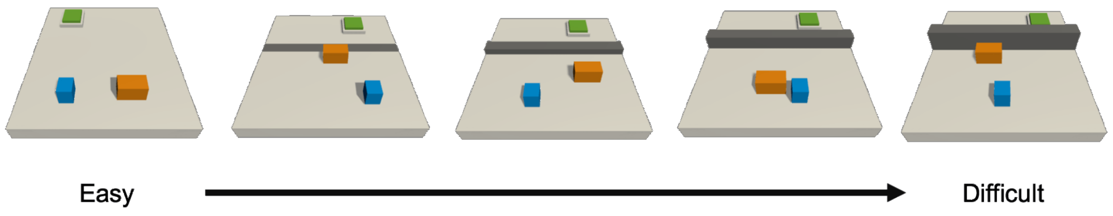


Figure 2.3: Demonstrate a curriculum training scenario where gradually a longer wall blocks the path to the target [6].

ing hyperparameters and data from 32 simultaneous agents. The difference is that the orange brain is trained using the full-size wall version of the task. The blue brain is trained using the task's curriculum version. The agent has many difficulties without using curriculum learning. After three million steps, orange agents still could not solve the task. Blue agents can accomplish much more complicated task with less time because of a well-prepared curriculum.

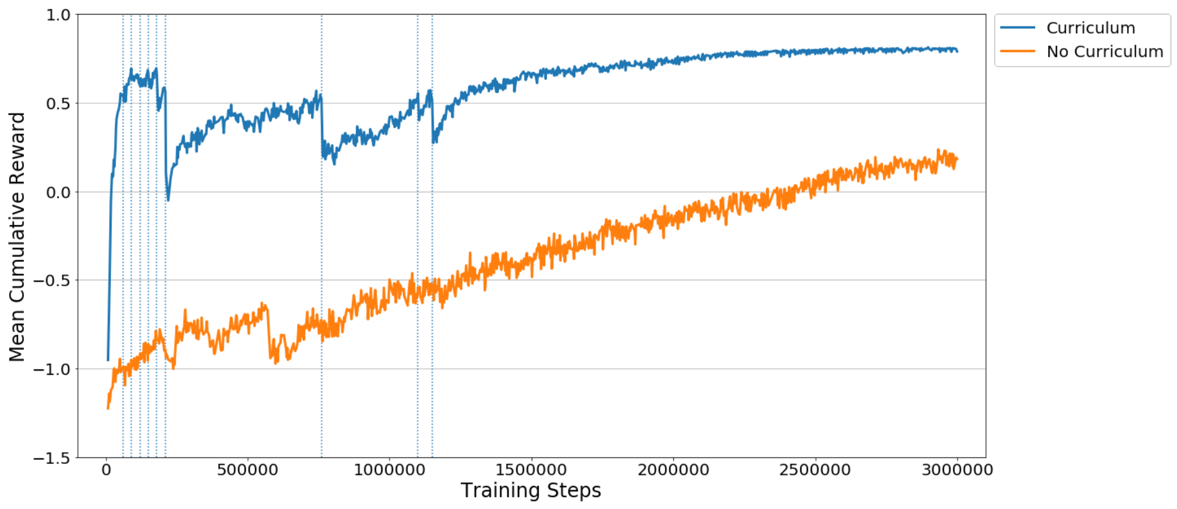


Figure 2.4: Cumulative reward of RL training session [6].

2.1.4 Hierarchical RL systems

Reinforcement learning provides an excellent solution if the reward is shaped and designed carefully. However, search-space is very large for a complicated task. Hierarchical Reinforcement Learning divides the complicated task to smaller pieces and manages tasks separately. A manager controls the subtasks and each subtask returns its current reward to its manager. To solve complex RL problems, HRL follows a "divide and conquer" approach. HRL provides suitable solutions for the followings:

Sample efficiency: The sample count will be very high when a complex task is desired to performed. It takes lots of time to evaluate all possible cases and find the best outcome. The main problem here is that search-space is huge when solving an integrated task. When an integrated task divides into subtasks, total search-space will be the sum of subtasks' search-space. For example, let the search-space of tasks A and B be 10 and 15, respectively. If the whole task is considered as integrated, the complexity of the entire task will be $10 \times 15 = 150$. If we divide the tasks into parts and evaluate each of them in itself, the search-space will be $10 + 15 = 25$. Thus, the search space used to solve the problem will be much lower than the total search-space. Considering more than two smaller tasks and the high search-space of these tasks, the search-space of the whole task would be too much. By dividing these tasks into pieces, both the tasks will be solved in a much shorter time, and more successful results will be achieved thanks to the limited search-space.

Generalization and transfer of learning: Every task performs a combination of small tasks. It is similar to use the letters to write a word. Although the number of letters is limited, the number of producing words is huge. Producing a behaviour or solving a integrated problem with RL is also similar to producing words. Although each task is unique when considered individually, it consists of sub-parts that make up itself. For example, many similar sub-tasks are used, when wearing clothes or putting clothes in a bag. There are sub-tasks such as walking, balancing, choosing the appropriate clothes and grasping the clothes. Once these subtasks are learned, they do not need to be learned again. If the clothes to wear are known, there will be only one additional task to be learned to put the dress in the bag. In this way, when the integrated task is divided into subtasks, previously learned behaviour could be transferred to a new complex task. The learned behavior will be applied to the new complex task as it is.

HRL [14] is working exactly like this situation. Several sub-policies work together in a hierarchical structure, instead of having just one policy that has to accomplish the goal of the whole task.

Feudal Reinforcement Learning

Feudal Reinforcement Learning (FRL) [15] describes a control hierarchy where a level manager can control their sub-managers, while super-managers control this level of managers. Each manager sets goals for his sub-managers, and sub-managers take actions to achieve this goal and get rewards.

FRL is based on two main principles: *Reward Hiding* and *Information Hiding*. Reward Hiding means that managers should reward sub-managers for making progress whether the super manager is performing commands. Thus, each manager learns to satisfy the higher level on his own. Information Hiding is referred to the fact that managers only need to know the state of the system in detail in their task selection. In this way, higher levels operate at a more extensive level of detail with a simplified state.

2.1.5 Unity ML Agents

Unity3D Machine Learning tool (ML-Agents Toolkit [16]) is an open-source machine learning tool project. It allows to design and develop smart agents for games, simulations and other applications. An agent can be trained by Reinforcement Learning, neuroevolution, imitation learning, and other machine learning techniques with its easy to use Python Application Programming Interface (API). It also provides applications of the latest algorithms (based on TensorFlow) which enables game developers, researchers, and hobbyists to train smart agents for 3D, 2D game and applications. These trained agents can be used for various purposes, such as controlling an NPC behavior, testing game structure, and deciding a preliminary version of the game's different structures. For both game developers and Artificial Intelligence (AI) researchers, Unity ML-Agents provides easy to use ML training environments, physical operations, visualizing, and others.

Unity ML-Agents consists of the following *Key Components* [7]:

Key Components

Learning Environment contains the Unity scene and all the game characters. The Unity scene provides the environment that agents can observe, act, and learn. According to the goals of the learning, Unity scene serves a custom learning environment. A specific reinforcement learning problem can be trying to solve a limited scope. Therefore, the same scene can be used for training and testing agents. Alternatively, training agents can be used to work in a complex game or simulation. In this case, it may be more efficient and practical to create a specialized training environment. The ML-Agents Toolkit includes an ML-Agents Unity SDK. It includes agents and behaviors, and it can transform any unity scene into a learning environment.

Python Low Level API - is a low-level Python interface for interacting and manipulating a learning environment. Phyton API, unlike learning environment, runs outside of the Unity and communicates with Unity via External Communicator. This API is included in the *special mlagents_envs* Python package. It is used by the Python training process to communicate and check with the Academy during training. However, it can also be used for other purposes. For example, API can use Unity as the simulation engine for other machine learning algorithms.

External Communicator links Unity Learning Environment with the Python Low-Level API. It is running as a part of the Learning Environment.

Python Trainers contain all the machine learning operations that enable training of the agents. Algorithms are implemented in Python, and mlagents is a part of the Python package. The package introduces a single command-line utility *mlagents-learn* that supports all the training methods and options supported by Unity ML-Agents Toolkit. Python Trainers only have interfaces with the Python Low-Level API.

The Learning Environment includes two Unity Objects used by Unity scene to organize [7]:

Agents are added to a Unity GameObject (any component in a game scene). It manages to create their observations, performing actions they receive, and assigning a reward (positive/negative) when appropriate. All Agents are connected to the Behavior.

Behavior is defined as a particular attribute of the agent, such as the number of actions it can take. A Behavior Name field uniquely identifies each Behavior. Behavior can be considered as a function that receives observations and rewards from the Agent and returns actions. Behaviors can be of three types: *Learning*, *Heuristic*, or *Inference*. Learning Behavior denotes the behaviour at the learning phase. Heuristic Behavior is

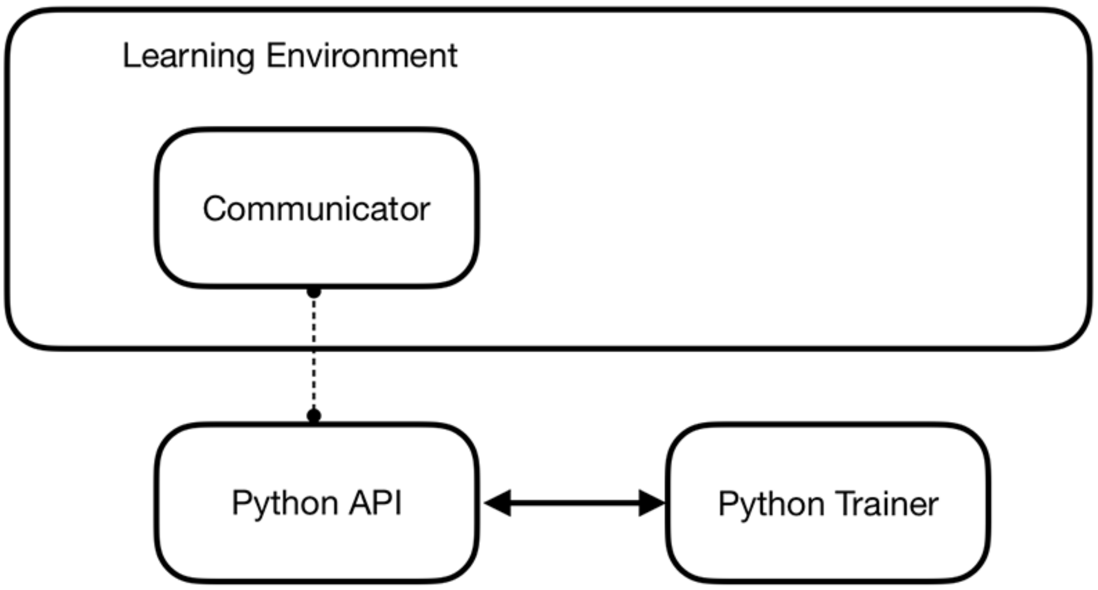


Figure 2.5: Simplified block diagram of ML-Agents. [7]

defined by a set of hard-coded rules applied in the code. Inference Behavior includes a trained neural network file. In reality, Learning Behavior becomes an Inference Behavior after trained.

All learning environments will always have an agent for each character in the scene. All agents are linked to behavior. Agents with similar observations and actions can have the same behavior. Thus, a learning environment can have two agents that do a similar task in the same manner. Of course, that does not mean that they will have the same observation and action values in all cases.

Hyperparameters are used by ML Agent Toolkit according to the trainer algorithm type [17]. These parameters determine the coefficients of the functions of the training algorithms.

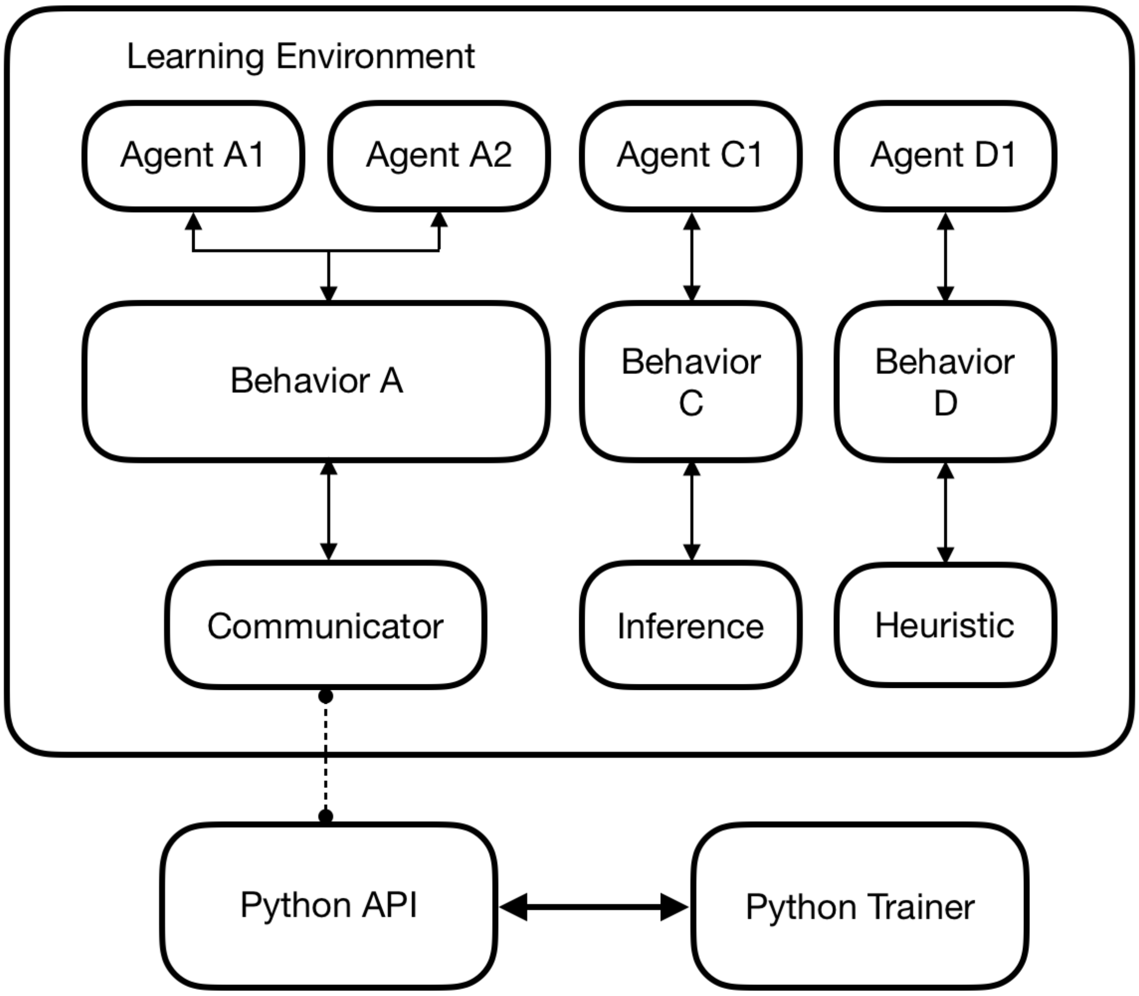


Figure 2.6: Simplified block diagram of ML-Agents [7].

2.2 Physics-based Animation

2.2.1 General Physics-based Animation

Animations are used to create more realistic and exciting content in applications, simulations and games. These animations are generally created previously and used by playing in the appropriate place within the application. This kind of pre-recorded animations are pleasing to the eye but will be insensitive to the environment since they are created for specific situations. They cannot create an animation according to the current state of the environment. With the laws of physics, realistic and pleasing animations can be produced automatically. The most significant advantage of physics-based animation is that it can produce realistic and integrated animations according to changing environmental conditions. However, doing physical calculations puts a massive strain on the processor. In order for an animation to look pleasing, it needs to be updated at least 30 times per second. The increase of physical objects in the scene makes it difficult to perform physical operations 30 times per second. For this reason, kinematic animations were used more in the early days. With increasing processor power and algorithm optimizations, limited physical animations are used today.

Rigid Body Simulation

The term rigidbody stands for the objects that do not deform under external disturbances. Rigidbody physics is one of the first techniques used for physics-based animation. In cases where two or more objects collide with each other, or when an external force is applied, the physical information of these objects is calculated. For a rigid body, only the position and rotation information is changed. Since the objects interact with each other, the calculation needs to be done iteratively more than once for more consistent results. More calculation steps produce more accurate results. Generally object with basic geometric shapes are preferred when performing rigid body simulation, because of performance concerns.

Soft Body Simulation

When force is applied on an elastic object, the object deforms accordingly. Soft body simulation intends to generate animations of such deformations. A soft-body object is generally obtained by establishing a spring-mesh structure. Many small rigid bodies interact with each other by being connected with this spring structure, which protects the initial shape of the object. When an external force is applied to the object, its shape changes as the springs push and pull the mass points. Besides calculating the collision of the mass points, the spring motion should also be calculated and as a denser spring-mesh structure is used, the soft body animation looks more realistic, which increases the computational burden on the processors. Nowadays, with the widespread use of physics libraries such as Havoc, Bullet, PhysX [18], physics operations can be calculated more optimized. Thus, it becomes possible to perform soft-body physics operations at 30Hz. Therefore, if we look at early applications, it is seen that rigid bodies are used more intensively.

Physics-based Character Animation

In most of the games, character animation consists of previously created animations. Keyframing and motion capturing are the main techniques for generating these animations. It is easier to make changes and optimizations on pre-recorded systems. For this reason, pre-recorded animations look better to the user. The main problem with pre-recorded animations is that animations are created for certain situations, so they cannot react to environmental forces. With physics-based character animation techniques, animations that can adapt to environmental changes are achieved [19]. The physics-based animation of the character with a combination of rigid and soft bodies and constraints between them can produce more natural results. In means of processing power, character animation consumes the most performance. For this reason, it is rare to see physical character animation in production games. Physics-based character animation is used in Max Payne 3 [20] and GTA V [21] for creating a more realistic environment and gameplay area. Besides, Ubisoft introduced motion-matching technology in 2016 [22]. The basic animations of the characters are recorded with mocap technology, and in real-time new animations compatible with the current game situation are synthesized by using them [23]. Although it produces results close to physical animation, completely realistic animation can not be generated because physics operations are not calculated.

2.2.2 PD Controller

PID Controller (denotes Proportional, Integral, Derivative, respectively) is a commonly used control loop mechanism in industrial control systems [24]. PID Controller is described as followin in [24]. Applications generally need a PID controller to regulate their value continuously. The purpose of this mechanism is to regulate continuous output via input. According to differences between input and output, proportional, integral, and derivative (PID, respectively) correction is applied to input value to reach output value.

In 1934, the PID Controller unit was placed on the market, and it is a widely used controller in process controller until today. PID controller is used for automated control of the process in the manufacturing industry. Also, besides mechanics, it is popular in the electronic controller. Nowadays, its commonly used for an application that requires precise, smooth, and optimized control mechanisms.

The closed-loop circuit systems are widely used in control systems. For example, consider a heating tank in which some liquid is heated to the desired temperature by burning fuel gas [24]. In closed-loop circuit systems, the current value needs to reach the desired value. The generated value is applied to the system (plant). The response of the system is sent back to the controller with the help of sensors. The controller updates itself according to the feedback, and the cycle continues in this way.

PID Controller is frequently used in physical animations and simulation applications. In computer animation, the integral component is not used to simplify the controller. Only the proportional and derivative components are used. For this reason, it is called PD Controller in computer animations.

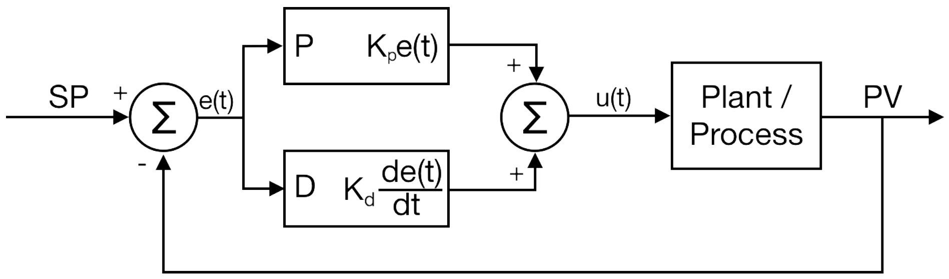


Figure 2.7: PD Controller Scheme

The *proportional* control is the easiest feedback control system. In proportional control, the signal of the controller is multiplied by the error signal. The optimal *Kp* value varies according to the system.

The *derivative* control calculates the time variation of the error in the system. Future changes are predicted from past changes. The effect of derivative control varies with the change of error by time. If the error increase rate is high, the effect of derivative control on the system is higher. If there is no change in error, it does not affect the system. The derivative control affects the output of the controller based on the change of error. Therefore, derivative control cannot be used alone. Besides, although derivative control is effective, noise is a problem for this controller [8].

Kp and *Kd* values can be varied according to the systems, and the best result can be obtained through trial. As shown in Figure 2.8, at different *Kp* and *Kd* values, the signal reaches the desired value in different ways.

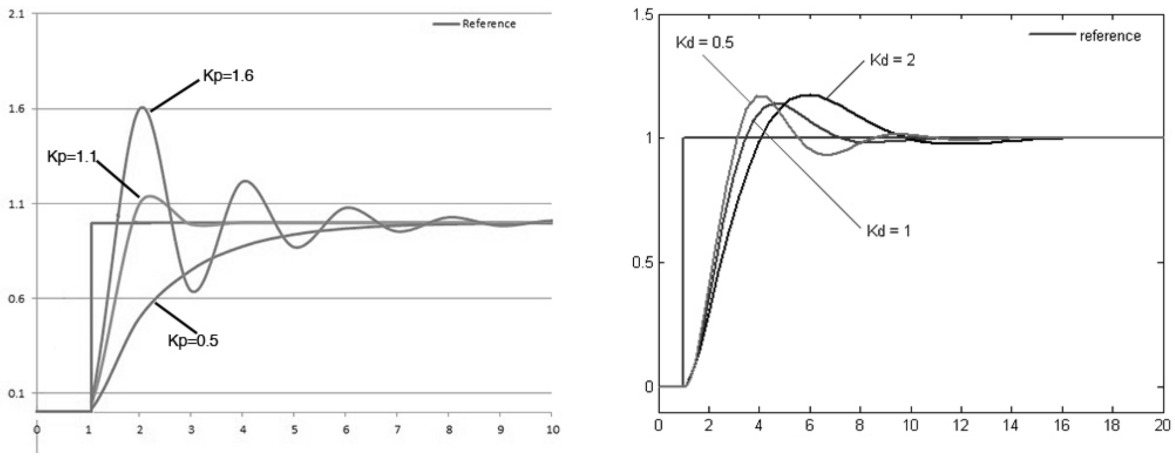


Figure 2.8: *Kp/Kd* sample chart [8].

3 RELATED WORK

In parallel to its tremendous role in real life, modeling the structure and motion of hands has been a popular topic of both animation and robotics research for decades. Kinematic techniques like key framing [25], or making use of anatomical or real hand pictures for generating 3D hand animations [26, 27] constitute the early approaches of hand animation research. During the last decades, hand animation research spread to a wide variety of fields like VR manipulation [28], modeling interaction with objects [29, 30, 31, 32] and, gesture generation [33, 34, 35], and recognition [36, 37, 38]. A comprehensive review on hand and finger modeling and animation can be found in the recent survey by Wheatland et al. [39]

In this study, we aim to synthesize physics-based catching motions, which can be categorized under hand-object interaction topic. We provide a detailed review of the literature on physics-based hand object interactions and their differences or similarities with our work in the sequel.

The early work on physics-based hand-object interaction synthesis are hybrid ones employing data driven techniques [29, 40]. Pollard and Zordan [29] present a physics-based controller for synthesizing hand motions like grasping and two-hand interactions by using motion capture data. They categorize the joint torques as active and passive and solve for the parameters of the controller from a set of motion examples. Their proposed method is applicable to new grasping scenarios with different object geometries. The used hand model is composed of rigid bodies but more detailed than common simplified models with additional degrees of freedom in the palm. Kry and Pai [40] also propose a dynamic retargeting system for grasping motion. They first capture the interaction of the hand with an object with a twofold process, they both capture the motion of fingers with an optical motion capture system and the contact forces with a force torque sensor attached below the surface. Then they resynthesize a new interaction of the hand with an object of different geometrical and physical properties. A recent data-driven approach is proposed by Zhao et al [41], for synthesizing grasping motion in real time with a hand model composed of rigid bodies. The key part of their framework is the online data-driven algorithm which synthesizes a grasping motion by using a large database of reference grasping motions. Once the proper motion is generated, the hand is controlled physically-in order to track this generated motion online.

Unlike these work, we aim to learn realistic catching motions from scratch, without relying on motion capture data, since using a small number of examples limit the variety of synthesized motions and on the other hand handling a large database of motion capture complicates the motion generation process. Another early work by Liu [42] presents an online physics-based manipulation controller which does not make use of motion capture data but need to be guided by simple descriptions of the desired motion. One of the interesting parts of her work is to solve the dynamics of the hand, the object and the contact forces in one procedure. The proposed framework, which controls both the arm and the hand, can be applied to a wide range of manipulation tasks. They also implement a more detailed model for the palm composed of rigid bodies.

Modeling the fingertips of the hand with soft body for increasing the stability of grasping has also been addressed in the literature [43, 44]. In the prominent work by Jain et al [43] the effect of employing soft body dynamics in the control of physics-based motions is investigated. According to their evaluations, modeling the deformation at the site of contact increase the robustness and naturalness of controllers for motions like locomotion, arm folding and pinch grasping a thin object. Inspired by their work, we modeled the deformable bodies as triangular meshes with point masses at the vertices and employed a spring based force calculation for simulating deformation. On the other side, they do not present a deformable hand model as detailed as we do, they only model the finger tips with soft body. More recently, Talvas et al. [44] proposed a novel contact constraint approach for simulating dexterous grasping of a variety of objects with soft fingers, including objects with sharp edges. They achieve an improvement on efficiency by aggregating multiple contact constraints into a minimal set of constraints. While they employ soft body modeling for fingertips, they do not model the rest of the hand including palm with soft body, which is essential for a stable grasping. Moreover they simulate the finger motions by tracking the motion generated by data glove or hand-scripted animations, while we aim to learn the proper finger motions from scratch. Our touch sensor model for improving the efficiency of training can be considered as a kind of aggregating contact points, which constitutes a similarity with their approach.

Including offline learning systems in hand motion generating approaches ease handling with the high dimensional and complicated nature of human motion. Andrews et al [45] propose a physics-based finger controller for one handed task based manipulation. They split the motions of the fingers for manipulation into three phases and learn a control policy for each phase offline with value iteration. They guide the learning process by narrowing the search space with natural poses. The hand model they use does not include soft bodies but they propose a more detailed palm model with a couple of rigid links. Our work is similar to theirs in point of devising an offline learning strategy for finger motion control. Cimen et al. [46] propose a controller for synthesizing a physics-based full body motion for catching, by dividing it into simple motor skills like standing, walking and reaching. Similar to our work, they employ an offline reinforcement learning system for generating a policy for the timing of these skills. Their proposed system can handle both single and double handed catching. Unlike ours, they do not include a detailed hand modeling and finger motion in their framework. Hao et al. [32] propose a physics-based controller for synthesizing motions of grasping and manipulating virtual objects with user guided multi-fingered hands in real time. The approach offers a general solution which is applicable to a wide variety of objects including high-genus objects with holes. Their algorithm includes a GPU accelerated offline learning phase for precomputing feasible grasp configurations for the given object. In real time, when the user-guided virtual hand is close enough to the object, the learned grasp space is searched for a nearest configuration and a physically plausible grasping motion, which is compliant with these configurations, is generated.

Incorporating a visual system imitating the function of human eyes for accomplishing high-level tasks like catching a thrown object is also an interesting research topic [1, 2]. Although it's not a physics-based approach, the work by Yeo et al [1] is a prominent one, in which movements of eye, head, hand and upper body during catching are all synthesized. They capture the motion of the body parts with motion capture and the motion of the eye with a head mounted tracker. They make inferences from these data about the synchronized movement of the body parts and eye, and build a model based on these inferences. They employ a simple visuomotor system and drive the movements of the bodyparts with the gaze information produced by this system. This work inspired us about handling the catching behaviour in two phases as reactive and proactive. Eom et al [2] propose a novel model predictive control framework for realistic eye, head and physics-based full body movements for various tasks including catching a thrown ball. Their key contribution is to enable interacting with the objects in a partially observable environment, where they model the observation of the environment with an integrated visuomotor system like the eyes of a human. Catching a ball is one of the motions that they demonstrate their proposed method. They divide the ball catching motion into reactive and proactive phases as we do, and adapt their framework parameters differently for these tasks. They calculate an estimated arrival position for the ball and produce an automatic lateral moving and arm lifting motion towards the arrival position. Unlike ours, their work does not include a hand animation. The main objective of our work constitutes the main difference from their work. We assume that the trajectory of the object is fully observable and focus on learning a detailed hand and arm motion for a stable catching.

4 HAND AND ARM MODEL

4.1 Hand Model

Human hand has a highly complicated structure which enables achieving a wide variety of motor skills with agility (see Figure 4.1). Modeling this structure with complex connection of bones, tendons and muscles is excessively hard and not efficient at all for animation applications. For these reasons, most of the work in the literature employ a simpler model of the hand for animation purposes [47, 48]. However, very simplistic hand models are not sufficient for generating realistic and stable motions in some cases; especially when interaction with objects is due. We propose a two-layer hand model, which enables a compromise between catching stability and performance.

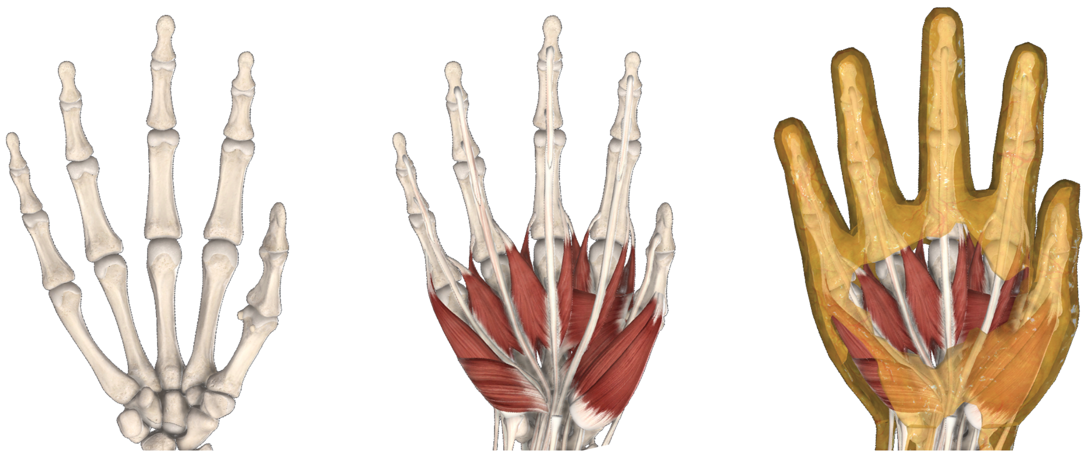


Figure 4.1: Hand anatomy: (a) skeleton, (b) skeleton with muscles and tendons, (c) skeleton with muscles, tendons, and fat [9].

The first layer of our hand model is a simplified physics-based skeleton model which comprises rigid bodies connected with joints. The structure of the skeleton model is displayed in Figure 4.2. The skeleton model consists of 28 Degrees of Freedom: one DOF for Distal Interphalangeal (DIP) and Proximal Interphalangeal (PIP) joints, two DOFs (flexion/extension, abduction/adduction) for Carpometacarpal (CMC) joint of the little finger and Metacarpophalangeal (MCP) joint except thumb , one DOF for Interphalangeal (IP) and Metacarpophalangeal (MCP) joints of thumb and three DOFs for Trapeziometacarpal (TMC) joint.

Usually, CMC joints are not included in simplified hand skeleton models [49, 41], but we observed that the movement of the little finger about CMC joint creates a better grasp by increasing the curvature of the hand. The CMC joints of other fingers have zero DOF in our model. The wrist of our hand model has three translational and two rotational DOFs. We model the hand such that its up axis is always aligned with the up axis of the arm and therefore we obtain the axial rotation of the hand by axial rotation of the arm. So wrist has only two rotational DOFs which are about x- and z- axes. In order to narrow the search space during the learning process, the maximum and minimum rotations of the joints are limited as displayed in Table 3.1. Moreover, the DOFs of DIP joints are modeled dependent on the DOFs of PIP joint by using the formula below, as proposed by Rijkpkema and Girard [50].

$$\Theta_{DIP} = \frac{2}{3}\Theta_{PIP} \tag{4.1}$$

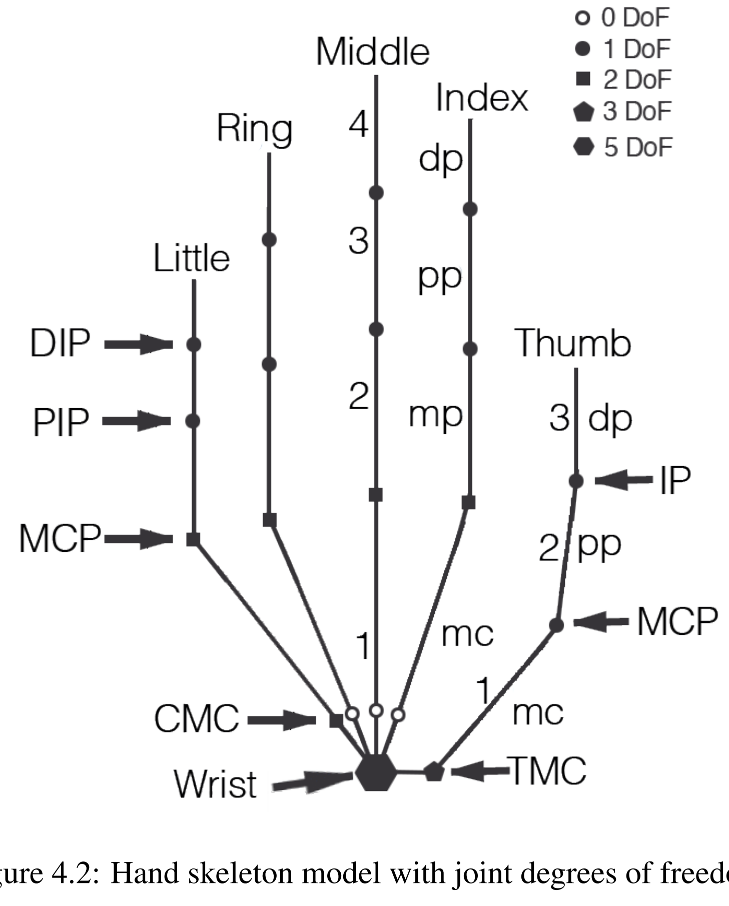
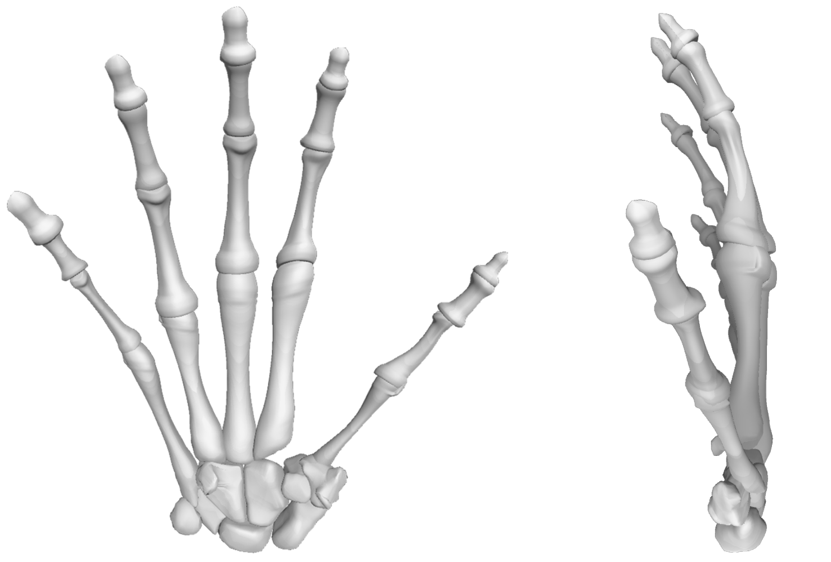


Figure 4.2: Hand skeleton model with joint degrees of freedom.

The movement of the finger rigidbodies is provided by applying torques to the joints. Given the desired orientation for a joint, the torque that will drive the attached finger to this orienta-

Table 4.1: Hand model properties. Finger joints do not have rotational degrees of freedom about the y axis [3].

Bone Name	Weight (gr)	Length (cm)	X Limit	Z Limit
Thumb1	35	4.92	0,60	0,60
Thumb2	24	3.41	0,45	0,0
Thumb3	16	2.21	0,70	0,0
Index1	72	7.11	0,0	0,0
Index2	28	4.39	0,90	-15,15
Index3	16	2.47	0,130	0,0
Index4	11	1.73	0,87	0,0
Middle1	62	6.90	0,0	0,0
Middle2	32	4.76	0,90	-15,15
Middle3	18	2.91	0,130	0,0
Middle4	12	1.85	0,87	0,0
Ring1	57	5.75	0,0	0,0
Ring2	28	4.45	0,90	-15,15
Ring3	18	2.94	0,130	0,0
Ring4	12	1.93	0,87	0,0
Little1	57	5.55	0,5	0,0
Little2	23	3.46	0,90	-15,15
Little3	12	2.02	0,130	0,0
Little4	11	1.76	0,87	0,0



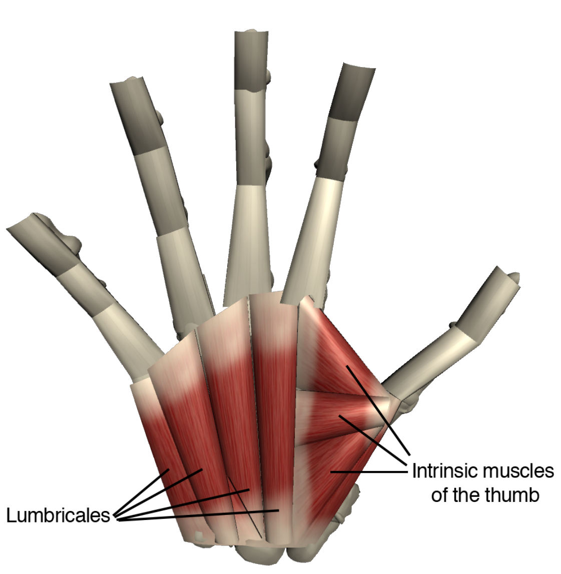
(a) Hand Skeleton Front (b) Hand Skeleton Side

Figure 4.3: Hand skeleton from front (a) and side (b)

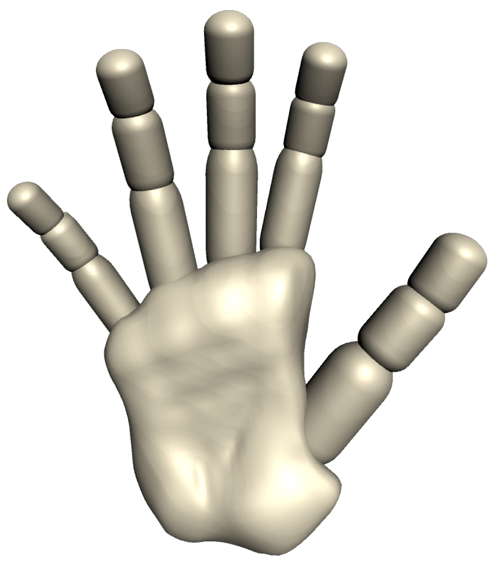
tion is calculated by PD Controller (see Section 2.2.2). In this way, a smooth physical movement is obtained. The base values for coefficients of PD Controller are set as $k_p = 331.293$ and $k_d = 7.098$ per each gram of the rigidbody. The coefficients for the finger rigidbodies are calculated by multiplying these base values with the weight of the bones.

The second layer of our hand model is a deformable model that intends to represent the soft tissue inside the hand. Human body is covered with fat, muscles, tendons and ligaments under the skin, which form a soft tissue deforming with the movement of bones or under the pressure of external perturbations. Representing the parts of the human body with deformable models, like hands [51, 52, 53], face [54, 55], or upper body [56], has been widely explored by researchers through decades, leading to a variety of approaches. Deciding which approach to use depends on the aim of the deformation modeling; it could be for aesthetic purposes, as in most of the facial animation problems, or for achieving a more stable and realistic interaction with the physical environment. Efficiency concerns are also important; do we need a real time render or do we prefer accuracy over efficiency? In our work, we aim a stable grip when the object is caught and we prefer the algorithm to run in real time. To this end, we first modeled a soft body layer both for palm and fingers, which is a simplification of real hand anatomy (see Figure 4.4a). Anatomically, the human hand has no muscles on the fingers. Instead the movements of the fingers are provided by tendons.

As a simplification, we modeled both the soft body covering inside the fingers and the palm with the same muscle-like structure.



(a) Hand with soft-bodies



(b) Hand simplified mesh

Figure 4.4: Hand with soft-bodies (a) and visual mesh (b)

Fingers have more prominent roles especially for the fine motor skills, like pinch-grasp, picking small items or writing; but for grasping during catching, the deformation of the palm is of great importance. We designed the muscles of the palm, inspired by the geometric muscles proposed by Albrecht et al. [52], but devised a simpler structure for performance concerns. We can categorize the muscles of the palm as intrinsic muscles of the thumb and the remaining four muscles called *lumbricales*. During catching, the fingers are bent towards the palm with the contact of the object. With this bending, lumbricales are curved and the thumb muscles press the object against them. We modeled the lumbricales and thumb muscles in different structures, which enables this cooperation for grasping the object (see Figure 4.4a).

We designed each muscle of the hand as a 3D triangular mesh in a certain geometrical shape, with elliptical cross sections through a designated range. A number of corner vertices of the muscle are determined as the attachment points, which attach the muscle to the bones of the hand. An attachment point is set as the child of the attached bone in the object hierarchy. Therefore, as the parent bone moves, the attachment points move with it, and the muscle deforms accordingly. Figure 4.5 displays the geometrical structure and the attachment points of palm muscles.

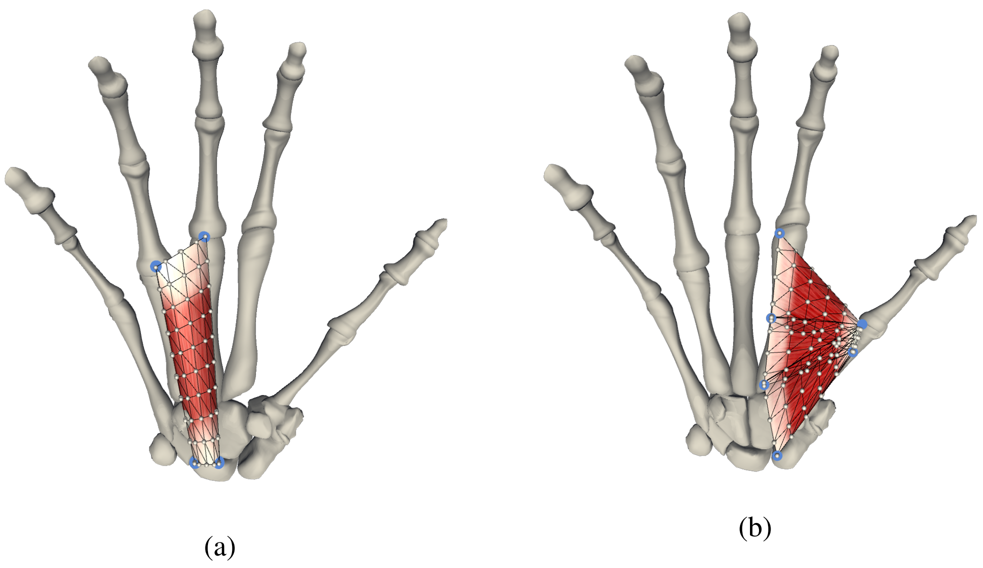


Figure 4.5: Muscle structure of (a) a lumbricale muscle and (b) intrinsic muscles of the thumb.

The contact of a soft body with an object causes a small and localized deformation [43]. We employed a simple and local deformation model for the physical interaction of the muscle mesh with an object as in [43] (see Figure 4.6). To this end, we defined a set of point masses and assigned the vertices of the muscle mesh as their rest positions. When an object collides with a point mass, a restoring spring force is calculated by the PD Controller with the intention of bringing the point mass to its rest position. The coefficients of the PD Controller define the stiffness and oscillation amount of the muscle under a collision. We manually set these parameters to k_p=100 and k_d=1.2, and we set the weight of the each point mass as one gram. The restoring forces calculated for the point masses of interaction push the interacted object towards each other and results in a more stable grasping of the object during catching.

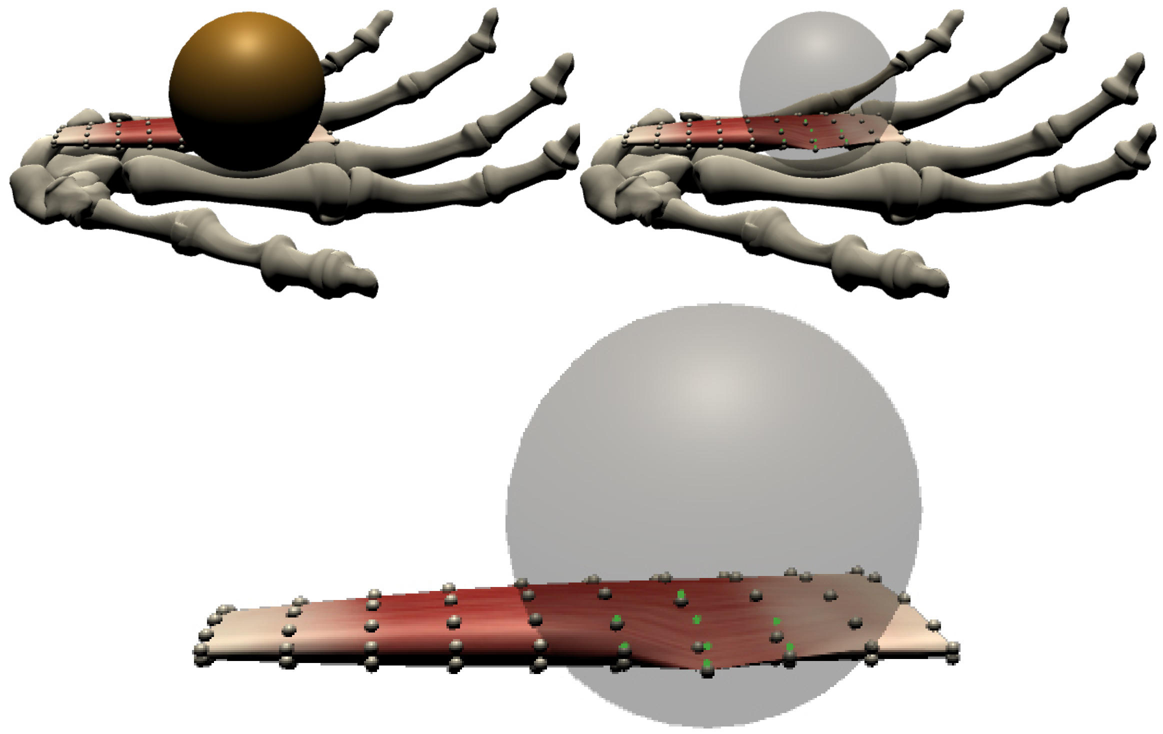


Figure 4.6: Deformation model for object-muscle interaction. Gray dots indicate the point masses which are positioned at the vertices of the muscle mesh by default, called rest position. When an object hits the muscle, point masses of contact displace with the effect of collision and a force is applied to restore them to their rest positions (green dots).

4.2 Arm Model

We used a simple arm model, consisting of upper arm, lower arm and hand, as shown in Figure 4.7. The lengths of the body parts and joint rotation limits are displayed in Table 4.2. Since it does not have an interaction with the environment, its movement is not physics-based. It rotates with the given rotation amount relative to its parent joint.

Table 4.2: Arm model properties [4].

	Joint	Body part	Length	X angle limit	Y angle limit	Z angle limit
1	Shoulder	Upper Arm	35	(-70, 150)	(-30, 90)	(-90, 90)
2	Elbow	Lower Arm	25	(0, 0)	(-135, 0)	(-90, 45)
3	Wrist	Hand	20	(-15, 15)	(-5, 5)	(0, 0)

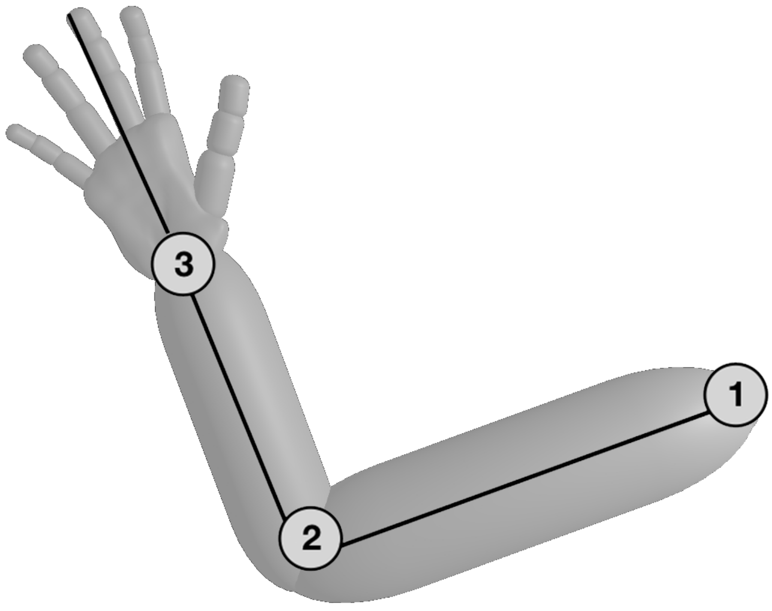


Figure 4.7: Arm model. Numbers indicate the joints in Table 4.2.

5 LEARNING TO CATCH WITH HIERARCHICAL REINFORCEMENT LEARNING

To produce a detailed caching animation, arm and hand should be animate according to each other. A catching process divided two parts; as the reactive and proactive phase. For successful catching operation, these phases should be run in coordination. Detailed information about the reactive phase, proactive phase and coordination manager of these phases is given below.

5.1 System Overview

We present the overview of our system, which is composed of an offline learning part and a real time motion generation part.

The offline learning system learns to control the arm and hand models so that a thrown object can be caught in a physically plausible manner. Catching motion is divided into two phases by Yeo et al. [1]. The first one is the reactive phase, in which humans get prepared for catching. This phase can include moving and orienting to a proper configuration, lifting the arm and keeping the hand at an estimated position until the thrown object gets close. The second one is the proactive phase, which consists of all the movements that contribute to carrying out the actual catching behavior. Accordingly, we partitioned our learning problem into two and devised a three-piece hierarchical learning system: the *reactive learner* for learning the reactive phase, the *proactive learner* for learning the proactive phase, and on top of these a third system, the *timing learner*, for learning the time management of reactive and proactive phases.

The reactive learner is designed to learn a natural motion of the arm for taking the hand to an appropriate position for catching. The system is trained with deep reinforcement learning (PPO).

The proactive learner is designed to learn the physical movements of the fingers for catching objects of several primitive shapes. The system is trained with curriculum learning.

The timing learner is designed to learn the timing of reactive and proactive phases of catching behavior, by using the trained brains by reactive and proactive learners. It is trained with deep reinforcement learning (PPO).

In real time, the state of the environment is given to the brain trained by timing learner (metabrain) and it gives the desired orientations for arm and hand joints as output, which are used to produce the whole catching motion.

Figure 5.1 displays an overview of the system. All of the learners in the offline learning system are based on reinforcement learning algorithms, which need the observations of the environment, action descriptions and reward signals as input. These input should be carefully designed to accomplish each desired goal. Detailed descriptions of the proposed learners are given in the following sections.

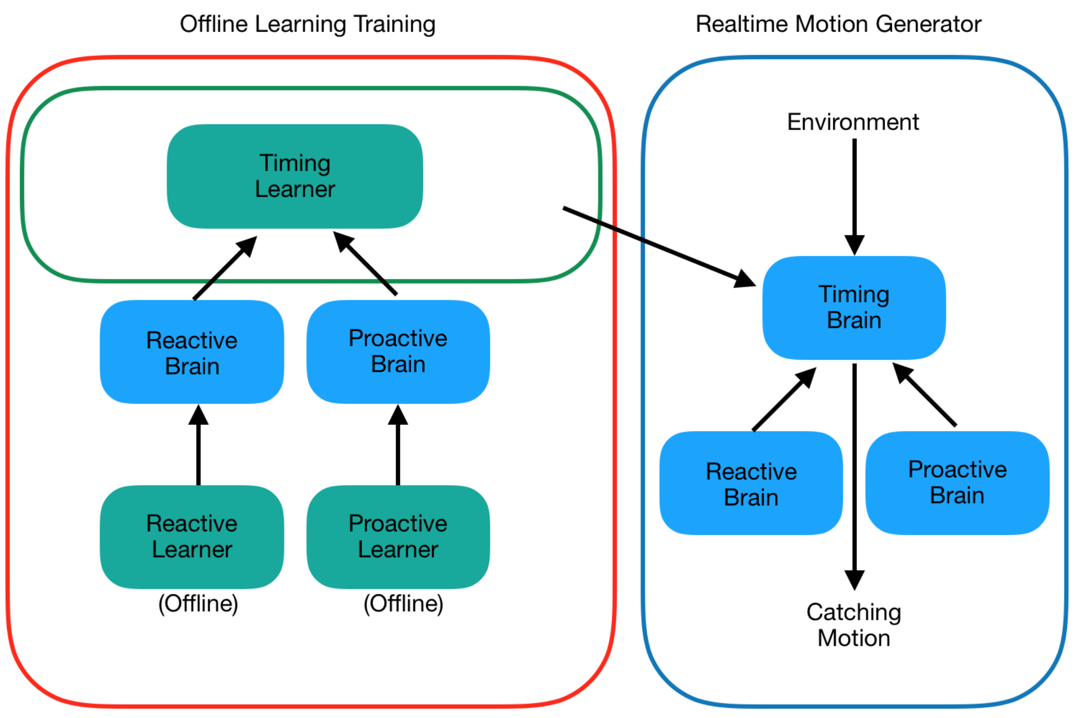


Figure 5.1: The system overview diagram. The system consists of an offline hierarchical learning system and a real time motion generator. The offline learning system is composed of three learners: reactive, proactive and timing learners. The timing learner uses the brains trained by reactive and proactive learners to train the metabrain. The metabrain is used as the main block of the real time motion generator system.

5.2 Reactive Learner with Proximal Policy Optimization

As mentioned in Section 5.1 reactive phase constitutes a preliminary preparation stage for catching the thrown object. Since we focus on generating only the arm and hand motions, this phase consists of placing the hand to an estimated interception position in our problem definition. Generating the arm motion for positioning the hand to a desired position is actually an inverse kinematics problem, which has more than one solution. Our aim is to learn an arm motion as natural as possible among all possible motions. To accomplish this, we employed Proximal Policy Optimization reinforcement learning algorithm [12] with the observation, action and reward design as follows.

Observation values:

- the position of arm parts,
- the local rotation of arm parts,
- the position of the object,
- the position of the hand palm,
- the distance of the object to the palm center,
- the velocity of the object,
- The accessible route: The part of the object trajectory that can be reached by hand is expressed as a five point curve,
- The position of the nearest two points to object from accessible route, and
- the forward vector of the hand.

Action values: A total of seven scalar variables are taken from the RL algorithm as action values. Each of these scalars correspond to an angular change in an arm joint's degree of freedom. Action values given by the RL algorithm are first clamped to [-1, 1] and then multiplied by three, so that at each step angular change is constrained to [-3, 3].

Reward design
The total reward for reactive learner is calculated as the sum of the total step reward (r_{restep}) and total termination reward ($r_{reterminal}$).

We designed the reward signals for calculating the total step reward in guidance of the following features of typical human behaviour, in order to achieve a natural arm motion.

1. Human tend to move their arms as little as possible.
2. The movement amount of the arm decreases as the object to catch gets closer.
3. They try to place their hand on the thrown object's course of movement.
4. They want to orient their hand such that the thrown object hits directly to the palm.

The total step reward consists of the following components, each of which is designed compliant with the features above:

- the displacement penalty,
- the route proximity reward, and
- the palm forward reward.

Displacement Penalty (r_{disp}): For the first and second feature we devised the displacement penalty, which gives a penalty for rotation of the arm joints according to the distance between the hand and the object.

We first calculate *dispPenaltyRaw*, which is a weighted sum of angular changes around each joint degree of freedom.

$$dispPenaltyRaw = w_s \times d(shoulder) + w_e \times d(elbow) + w_w \times d(wrist),$$

Here, function d calculates the sum of the angular change around the given joint's degrees of freedom. The coefficient for each joint should be compliant with the joint's effect on the displacement of the hand. Here we set $w_s = 2$, $w_e = 0.7$ and $w_w = 0.3$, because the effect of shoulder movement is the largest and the wrist's effect is smallest on hand displacement.

We calculate the displacement penalty r_{disp} as below. Here *dist* denotes the distance between the object and the hand and *maxDist* denotes the maximum value of *dist*. With this equation, the penalty for displacing the hand gets larger as the object gets closer, which is compliant with the second feature of typical human catching behavior.

$$r_{disp} = (1 - \frac{dist}{maxDist}) \times dispPenaltyRaw. \tag{5.1}$$

Route Proximity Reward (r_{route}): For the third feature we devised the route proximity reward with the following *distRoute*. Here *distRoute* denotes the closest distance between the palm and the route of the thrown object. With this formula, the closer the palm is to the route, the more reward is given. If the distance is greater than 30 cm, then no reward is given.

$$r_{route} = (e^{((1-\frac{dist}{distRoute})\times 3)} - 1)/(e^3 - 1)$$

Palm Forward Reward ($r_{forward}$): For the fourth feature, we devised the palm forward reward with the following formula, which rewards the alignment of the velocity of the thrown object (*objectVel*) with the forward vector of the hand (*handForward*). Here the *angle()* method calculates the least unsigned angle between two vectors.

$$r_{forward} = (e^{((1-\ (180-angle(objectVel,handForward))/180)\times 3)} - 1)/(e^3 - 1)$$

The total step reward value for the reactive learner is calculated as the weighted sum of these three rewards:

$$r_{restep} = w_1 \times r_{disp} + w_2 \times r_{route} + w_3 \times r_{forward}. \tag{5.2}$$

Here, we set $w_1 = -0.1$, $w_2 = 0.35$ and $w_3 = 0.65$, which we tuned according to the training results.
The termination reward is calculated as

$$r_{reterminal} = r_{palmhit} + r_{pos}.$$

Palm Hit Reward ($r_{palmhit}$): The purpose of the movement of the arm is to collide the thrown object with the palm. In this case, the training is completed with success and awarded +1 point.

Object Position Penalty (r_{pos}): The position of the object must be inside certain limits around the hand and arm in order to catch the thrown object. The training is terminated when the object gets to a position that the arm cannot reach or when the object passes behind the shoulder. Since the training is terminated unsuccessfully in these cases, -1 point is given.

5.3 Proactive Learner with Curriculum Learning

In the proactive phase of catching motion, humans aim to close their fingers in the correct time and configuration in order to grasp the thrown object in the most stable way. The orientation of finger joints at the final grasping pose and the speed of closing the fingers through this orientation are determined according to the shape, size and velocity of the thrown object. In this part of our work, we aim to learn the best possible physics-based catching motion for the hand model with muscles described in Section 4.1. The thrown objects can have a variety of shapes with primitive geometries like sphere, cube, capsule and rectangular prism (see Figure 5.2).

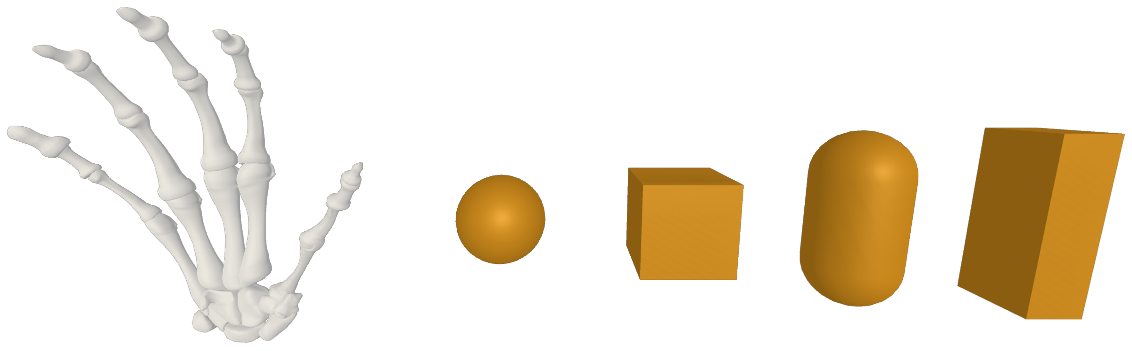


Figure 5.2: Hand skeleton and thrown objects.

Some objects are harder to catch than others, mostly due to their shapes. Objects with uniform geometry are easier to grip, while the smoothness of the object is also a factor. Considering this, we designed a gradual training, in which the hand learns to catch objects with simpler geometry former than the others and becomes more talented as the object geometries get more complex. We achieved this by employing curriculum learning algorithm explained in Section 2.1.3. As shown in Figure 5.3, training starts only with the objects of simplest geometry, spheres, and other objects are included in the training later, at certain time steps, according to the complexity of their geometry. In this way, by expanding the search space step by step, faster and better results are achieved.

Curriculum learning is based on reinforcement learning. The observation, action and reward design for the underlying RL algorithm are as follows. For the observation and reward design, we propose a hand model with sensors on it at specified positions, as shown in Figure 5.4. The sensors are sphere colliders, and for each of the sensors a touch status information is hold. The touch status of a sensor is set as active during the time it collides with the object and set as passive vice versa. Therefore, it is possible to estimate the grasp amount of the object by interpreting the touch status of the sensors, which we use for reward design. The grasp estimation process is described in detail in the sequel.

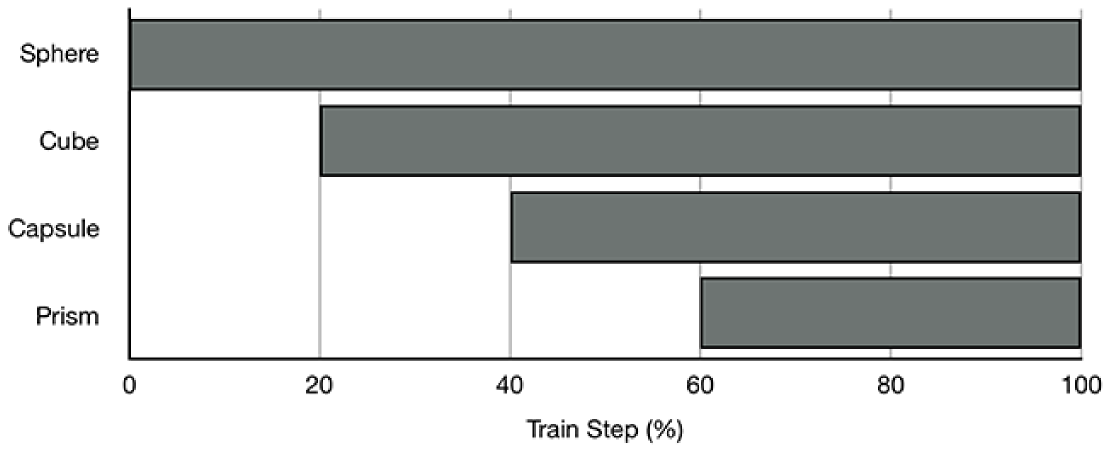


Figure 5.3: Curriculum learning timeline of the used object shape.

Observation values:

- the rotation of the hand,
- the local position of each finger part,
- the local rotation of each finger part,
- the velocity of each finger part,
- the angular velocity of each finger part,
- the touch status of each sensor inside the hand (Total 28 sensors),
- the touch status of each sensor outside the hand (Total 7 sensors),
- the ifference vector between the object position and the hand position,
- the velocity of the object,
- the rotation of the object,
- the angular velocity of the object.
- the distance between the object and the hand.
- Shape type of the object (an integer between 0 and 3).

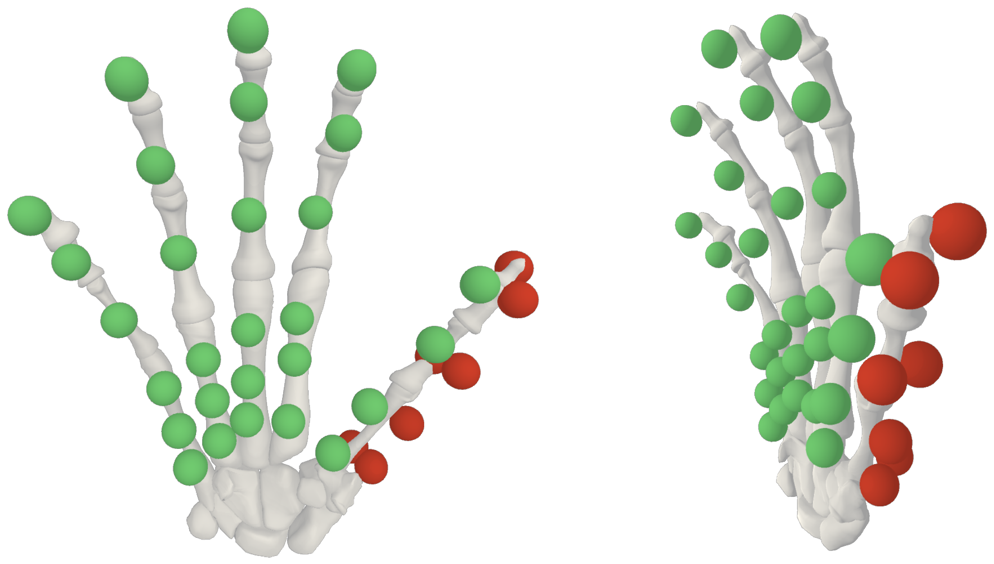


Figure 5.4: Sensor positions on the hand. There are 28 inside sensors (green) and 7 outside sensors (red).

Action values:

Each action value corresponds to an angular change at the desired orientation of a finger joint. As seen in Figure 4.2, our hand model has 23 degrees of freedom for finger joints. Taking into account the angular relation between DIP and PIP joints (see equation 4.1), we take 16 scalar variables for the RL algorithm as action values. These action values are first clamped to [-1,1] and then multiplied by a strength value of 60. Here, we prefer a larger strength value, since the finger movement should be faster compared to arm and the action update frequency is smaller than the reactive learner (see Section 5.1). For some degrees of freedom these processed action values are directly added to the desired angle value while for others, whose movement is restricted to only positive direction, absolute values of them are used. After adding these values to desired rotation angle, the resulting angle value is clamped to a specified interval as displayed in Table 5.1.

Reward design

As mentioned above, we devised a touch sensor based reward calculation to measure the grasping amount and duration of the hand. There are 28 sensors inside and 7 sensors outside of the hand. Since the object is always thrown from the front of the hand, we didn't place sensors outside the fingers except thumb. Exterior of the thumb can touch the object if it's

Table 5.1: The absolute value of processed action values are added to some DOFs, where the movement is restricted to only positive direction. The total rotation after adding action value is clamped to a specified interval.

Add Absolute	DOF	Clamp Interval
	Thumb1 [Z]	[-20,40]
✓	Thumb1 [Y]	[0,Thumb1[Z] + 20]
✓	Thumb2 [X]	[0,45]
✓	Thumb3 [X]	[0,70]
✓	Index2 [X]	[0,130]
	Index2 [Z]	[-15,15]
✓	Index3 [X]	[0,130]
✓	Middle2 [X]	[0,130]
	Middle2 [Z]	[-15,15]
✓	Middle3 [X]	[0,130]
✓	Ring2 [X]	[0,130]
	Ring2 [Z]	[-15,15]
✓	Ring3 [X]	[0,130]
✓	Little1 [Z]	[0,5]
✓	Little2 [X]	[0,130]
	Little2 [Z]	[-15,15]
✓	Little3 [X]	[0,130]

rotated in the wrong time with a wrong amount. Therefore we needed to place sensors there. The locations of the sensors are displayed in Figure 5.4.

The total reward for proactive learner is calculated as the sum of the total step reward ($r_{prostep}$) and total termination reward ($r_{proterminal}$).

The total step reward is composed of the following components :

Inside sensor reward (r_{inside}): The number of active sensors inside the hand are strongly correlated with the status of the catching. If no sensors are touched by the object, this means that the object is not grasped by the hand. As the number of active sensors increases, we understand that the object is grasped more firmly and it's more unlikely to drop it. Therefore inside sensor reward is one of the most dominant reward components. There are total of 28 inside sensors, but since the objects are rigid, we observed that even in a full stable grasping all of these sensors do not get active. But according to the shape and angle of incidence of the object, which sensors get active may vary. Therefore, we need all of the 28 inside sensors for interpreting the catch status of different objects, but the maximum number of active sensors vary between 10 and 15. Because of that, we first calculate the $norm_{active}$ value by dividing the total number of active sensors by 14 and then clamping the resulting value between 0 and 1 to get a normalized ranking for the grasping amount. The inside sensor reward r_{inside} is then calculated as follows:

$$r_{inside} = (e^{3 \times norm_{active} - 1}) / (e^3 - 1)$$

Inside thumb sensor reward/penalty (r_{thumb}): Covering the object with thumb is essential both for a more stable catching and a more natural motion. Therefore, we include an additional term, r_{thumb}, for rewarding the sufficient contact with the interior of the thumb and penalizing vice versa, calculated as follows:

$$r_{thumb} = (num_{active} - 2)/2.$$

Here num_{active} denotes the number of active sensors inside the thumb. As clearly seen by the formula, r_{thumb} takes a negative value if num_{active} is less than 2 and a positive value vice versa.

Outside sensor penalty ($r_{outside}$): This reward item stands for penalizing the wrong movement of the thumb, such that the object contacts with its exterior. There are 7 sensors outside the thumb, shown as red in Figure 5.4. The total number of active ones is divided by 7 to calculate a normalized reward value, $r_{outside}$.

The total step reward $r_{prostep}$ for the proactive learner is calculated as follows, as the weighted sum of these three rewards:

$$r_{prostep} = w_1 \times r_{inside} + w_2 \times r_{thumb} + w_3 \times r_{outside}.$$

Here we set $w_1 = 0.7$, $w_2 = 0.3$ and $w_3 = -1$ which we tuned according to the training results.

The total termination reward $r_{roterminal}$ is calculated as:

$$r_{proterminal} = r_{distance} + r_{steady},$$

where r_{steady} and $r_{distance}$ are described as follows:

Object distance penalty ($r_{distance}$): When the distance between the object and the hand is more than a specified value, a penalty of -1 is given and the training episode is terminated. This penalty stands for penalizing the case that the hand cannot catch or drops the object. Here we specify the distance value as 24 cm, since the object is thrown from 24 cm at most. Since the object is thrown towards the hand, it means that the hand could not catch the object or dropped it if it moves away from the hand this far.

Steadiness reward/penalty ($r_{textitsteady}$): For achieving a steady catching behavior, we require the hand to grasp the object just not for an instant but for a while. Therefore we reward the case that the hand still touches the object at the end of the training episode and penalize vice versa. For this, in the last step of the episode, if at least one of the inside sensors is active, then a reward of $+1$ is given. In the opposite case, a penalty of -1 is added to the total reward.

5.4 Timing Learner with Hierarchical Deep Reinforcement Learning

Catching is accomplished with the collective motion of different body parts, each of which is responsible of a distinct task. The timing of these distinct tasks is of great importance, since the object to catch is not stationary, on the contrary it's in motion most probably with a high velocity. When we consider generating a catching motion as a learning problem, it's also quite plausible and natural to split the motion into different phases. As we mentioned above, we split the problem into two, as learning reactive and proactive phases separately, and we designed learning the timing of these different tasks as a hierarchical reinforcement learning problem (see Section 2.1.4). The relation between timing learner and the trained brains by reactive and proactive learners are displayed in Figure 5.5. The working principle of timing learner is briefly described as follows. The object is thrown from a random distance between 200-300 cm towards the character. The timing learner sends a run command to the reactive brain at a time specified by an action signal coming from the reinforcement learning algorithm. When the object approaches the hand closer than 24 cm, the timing learner sends a run command to the proactive brain at a time specified by another action signal coming from the reinforcement learning algorithm. The reactive and proactive brains follow their learned strategy and send reward signals described below to the timing learner at each step. In addition, the proactive learner sends a termination signal to the timing learner, when the termination criterion for an episode is provided. The details of observations, actions and reward design are described below.

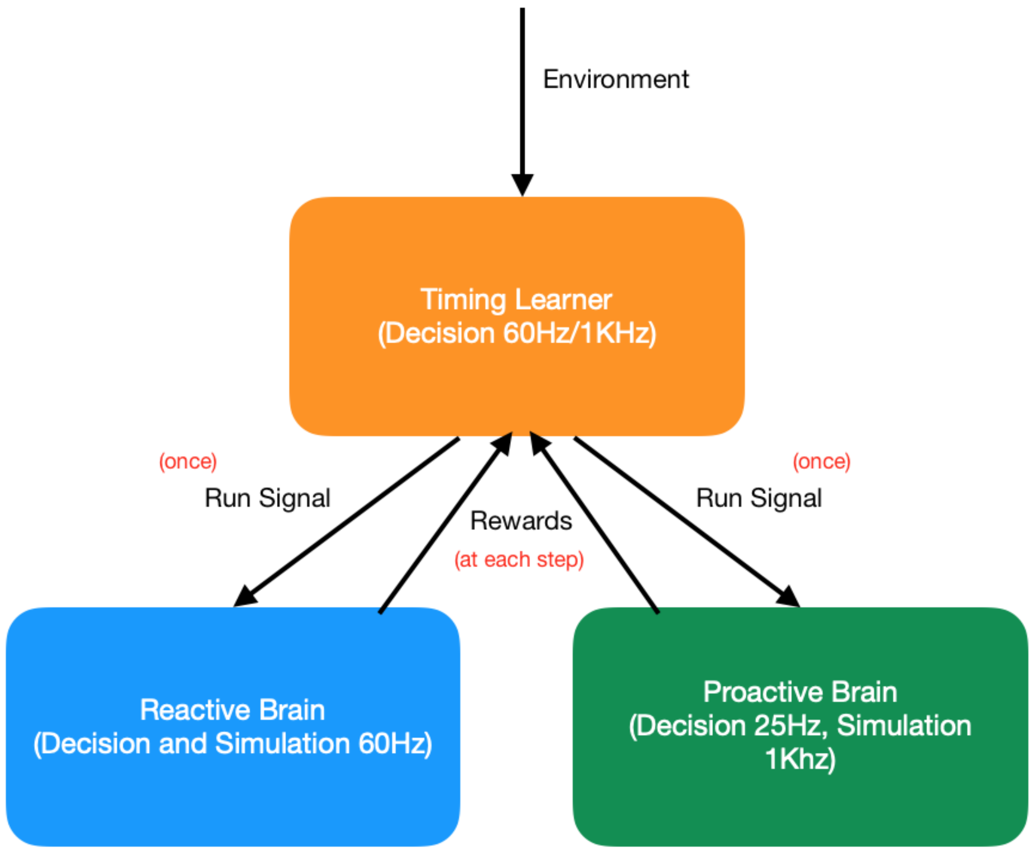

Environment
Timing Learner (Decision 60Hz/1KHz)
(once)
Run Signal
Rewards
(at each step)
(once)
Run Signal
Reactive Brain (Decision and Simulation 60Hz)
Proactive Brain (Decision 25Hz, Simulation 1Khz)

Figure 5.5: The relation of timing learner with reactive and proactive brains. Timing learner sends the "start running" command to them once. The brains send the specified reward values at each decision time step.

Observation values:

- the shape type of the object,
- the distance between the object and the palm,
- the difference vector between the object position and the hand position,

- the velocity of the object,
- the success status of the reactive brain,
- the success status of the proactive brain,
- the hand touch status (is object-hand distance less than 15 cm), and
- the hand inside touch status (is object-hand distance less than 5 cm).

Actions:

- send "start running" signal to reactive brain.
- send "start running" signal to proactive brain.

Reward Design

The total reward for timing learner is calculated as the sum of the total step reward (r_{step}) and the total termination reward ($r_{terminal}$).

The total step reward is calculated as:

$$r_{step} = 0.5 \times r_{reactive} + 0.5 \times r_{proactive},$$

where $r_{reactive}$ and $r_{proactive}$ are described as follows:

Total step reward of reactive brain $r_{reactive}$: After the reactive brain runs with the run command coming from the timing learner, at each step the total step reward of reactive brain (r_{restep}) is taken.

Total step reward of proactive brain $r_{proactive}$: After the proactive brain runs with the run command coming from the timing learner, at each step the total step reward of proactive brain ($r_{prostep}$) is taken.

The termination reward is calculated as:

$$r_{terminal} = r_{repos} + r_{protime} + r_{prosteady},$$

where r_{repos}, $r_{protime}$, and $r_{prosteady}$ are described as follows:

Object position penalty r_{repos}: This reward is the same as the one in reactive learner reward design (r_{pos} in Section 5.2) and taken from the reactive brain. The training is terminated if the object passes behind the shoulder or falls below the height at which the arm can reach, and a penalty of -1 is given.

False proactive timing penalty $r_{protime}$: The proactive brain should run when the object-hand distance is between 16 and 24 cms. If it runs out of this interval, the episode is terminated and a penalty of -1 is given. Moreover, if the proactive brain has not run yet when the object hits the palm, then again the episode is terminated and a penalty of -1 is given.

Proactive steadiness reward/penalty $r_{prosteady}$: This reward is the same as in proactive reward design (r_{steady} in Section 5.3) and taken from the proactive brain. It is calculated in the last step of the episode and valued +1 if at least one of the inside sensors of the hand is active and -1 vice versa.

6 IMPLEMENTATION AND RESULTS

This chapter covers the implementation details of the proposed system and the details about the resulting simulations. Moreover, we provide a comparison of the results with different configurations of the framework and user tests conducted for evaluating the results are also presented in this section.

6.1 Implementation Details

Offline machine learning algorithms and resulting online simulations were all performed on a 2.70 GHz 4-core machine with 16GB of memory. Unity 3D game engine version 2019.1.171f was used for rendering and animation [57]. Nvidia Physx SDK 4 [58] is used as the black box physics simulator, which is the built-in 3D physics engine of Unity. Unity ML Agents v0.12 is used as the machine learning environment [59]. Since our implementation includes physics simulations and Nvidia PhysX runs on CPU, parallel processing with GPU could not be used for accelerating offline learning.

In the experiments, the hand and arm models described in Section 4 have been used. As mentioned in Section 5.3, four basic types of objects were used in the simulations. The shapes with regular mesh geometry have been intentionally preferred because of their efficiency in collision handling. The objects sizes are constant, which have been adjusted so that they can be covered by the hand easily (see Figure 5.2).

Proactive Learner

Hand motion is generated with physical simulation with integration time step of 0.001. Therefore, the training environment for proactive learner runs at 1 kHz. Moreover at each physics time step, joint torques for 23 DOF hand model and deformation forces for point masses of the soft bodies are calculated. These physics calculations all cause a CPU overprocessing during proactive learning. To achieve a CPU optimization, the training environment has been adjusted with some initial simplifications. The object is positioned initially just across the hand, in a random distance between 16 and 24 cm, facing the palm. The hand is first oriented to the optimal orientation for catching, such that the up vector of the object is aligned with the vector normal to the plane formed by the thumb and index finger. After that, its orientation is randomized by rotating it about z-axis with a random amount between [-20, 20] degrees and about x-axis with a random amount between [-40,40] degrees. The initial configuration of the hand is displayed in Figure 6.1. With these simplifications, the search space of the learning algorithm is narrowed, which enables exploring the main search area as the motion space of the fingers. In this way, the training time is reduced.

Although high frequency is needed for physics-based calculations, a lower frequency can be used for the decision making process of learning algorithm. We set the frequency as 25hz.

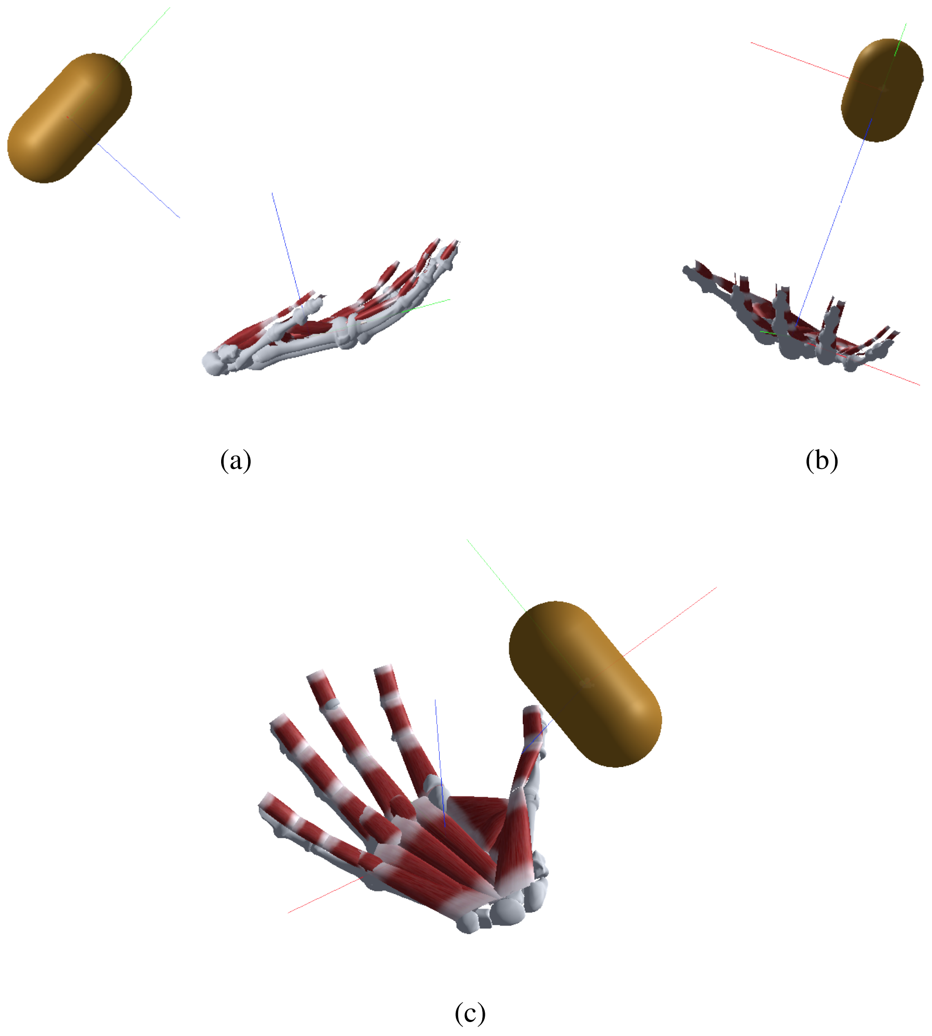

(a) (b) (c)

Figure 6.1: Proactive learner environment is shown from different angles.

Reactive Learner

Unlike hand, arm is both modeled and controlled kinematically. We did not opt for using physics-based methods for arm control because of several reasons. First of all, the main focus of this work is to generate realistic hand motions for grasping the object in a stable way during catching. The arm motion is considered as an auxiliary motion to take the hand to the desired position and orientation. Besides, we do not intend to model the object-arm collisions. For these reasons, we modeled the arm motion as simple as possible for achieving a faster training. The reactive learner works at 60 hz.

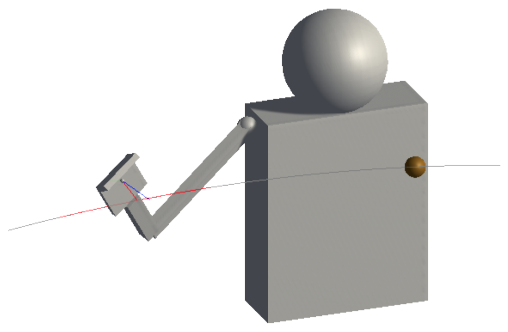
As in the proactive learner, we made some simplifications and assumptions about the training environment. We do not want to include scenarios where the objects are thrown at the speed and directions that the hand cannot catch. Therefore the target positions and speed of the thrown objects are restricted.The object is thrown to the character at a distance of 2 to 3 meters from the head (the gray box in Figure 6.2). The initial velocity of the object is set as 2 m/seconds. The object can be thrown to the character from all directions, except towards the bottom left of the character, since it is quite difficult to capture an object with right hand in this area. The area around the character where the objects can be thrown is displayed in Figure 6.3. In reactive learner, we use a simple hand model, which is used only to detect if the object collides with the hand (see Figure 6.2).

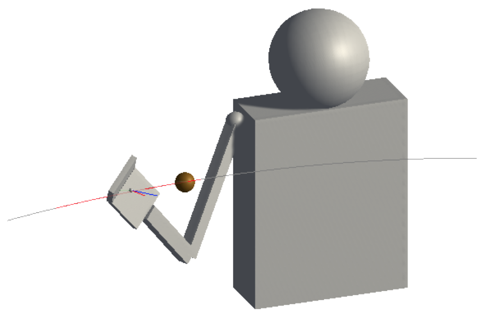
Timing Learner

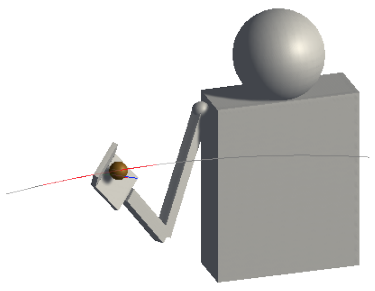
As explained in Section 5.4, timing learner takes the previously trained brains by reactive and proactive learners. Therefore, the training environment restrictions for these learners, also apply for the timing learner. Figure 6.3 displays the training environment with mentioned restrictions.

Reactive and proactive brains have different decision and simulation frequencies. Reactive brain aims to animate the kinematic arm model, and it both updates and makes decisions, produces actions at the frequency of 60 Hz. On the other side, the proactive brain controls the physics-based hand model and makes decisions at 25 Hz frequency and for the sake of stability physics simulations run at 1 kHz. For carrying out the training as fast as possible, we designed a variable frequency training environment for the timing learner as follows: Before the proactive brain starts to run, the decision and simulation frequencies are set to 60

Hz. As the proactive brain starts running, the decision and simulation frequencies are set to 1 kHz. In this way, we optimize the timing learner by adjusting the simulation and decision frequencies according to the model and simulation type.


(a) The object is thrown and the hand moves to the target.


(b) The hand is positioned on the accesible route shown with red.


(c) The object hits the center of the palm.

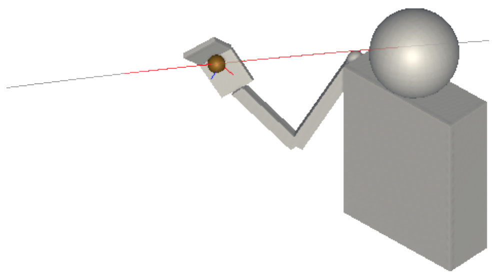
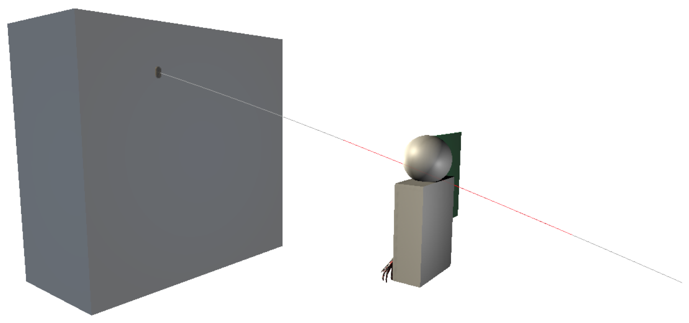
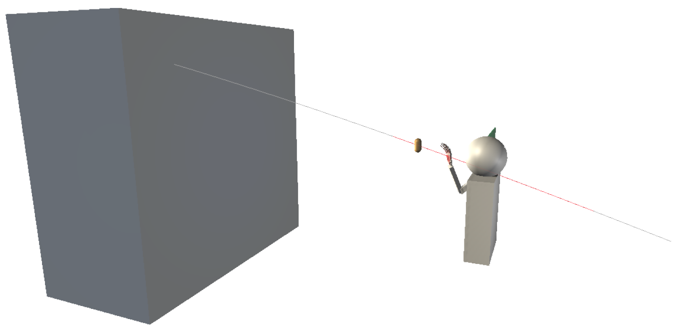
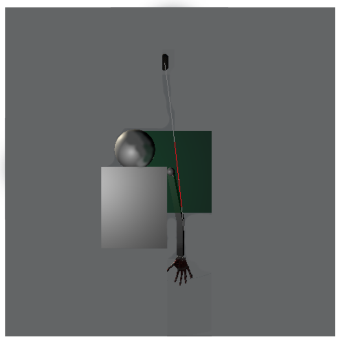

(d) The collision from another angle.

Figure 6.2: Some key steps from the Reactive Learner environment. Note that the hand is modeled as a simple articulation composed of two rigid bodies in this environment.


(a)


(b)


(c)

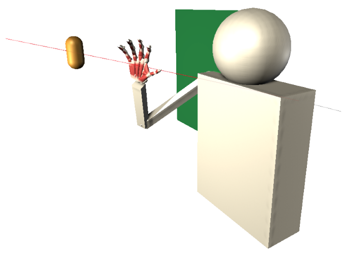

(d)

Figure 6.3: Some instants from the Timing Learner environment. The gray box in (a) and (b) denotes the region for the initial position of the object. The green region in (c) and (d) is the area where the objects can be thrown. Note that the hand model is the detailed one in the Proactive Learner.

6.2 Training Parameters

All of the offline learning systems in our work are trained with Proximal Policy Optimization (see Section 2.1.2). Proximal policy optimization is a deep reinforcement learning algorithm developed for the training of policies with continuous actions, as in our problem.

Table 6.1 displays the hyperparameters and other training information for the machine learning environments in our framework. The explanation of PPO hyperparameters in Unity ML-Agents Environment can be seen in [17]. The *MaxStep* parameters vary depending on the agent's condition and have been determined by trial and error. The beta parameter is used to increase the randomness in training; we used a larger value than the default value. The *lambda* value has been used as small as possible because the actions also affect subsequent states. *hidden_unit* is set as the closest value to the number of observations. Rewards are given to the trainer by normalizing them and it gives continuous action values as output. Other parameters are set to their default values.

Along with hyperparameters, other information used for training is also displayed in Table 6.1. The training durations are adjusted by trying to achieve a compromise between a successful training and performance.

Tensorboard charts of training can be seen in Figure 6.4, 6.5, and 6.6 for the reactive, proactive and timing learners, respectively. In these charts, it can be seen that the total reward increases over time in all of the trainings. The episode length differs depending on the type of training. In the reactive learner chart, it reaches the object in less time as it learns the environment. However, the timing learner can do its job better as it learns the environment. Because of that, their episode length trend is not the same. *Lesson* means the stage of curriculum learning. Only proactive and timing learner use this value. Lesson value and learning environment complexity increased over time for proactive and timing learner.

Entropy is the randomness of the policy which is generally expected to decrease over time. As seen from the charts, *Entropy* values for the reactive and proactive learners decrease over time. However, it starts to decrease and then increase for the timing learner. This increase point corresponds to the time when the process with reactive brain is complete and the proactive brain starts to run. Since the values like episode length and decision frequency

Table 6.1: Training parameters

Parameters	Proactive Learner	Reactive Learner	Timing Learner
Training duration	60h	24h	72h
Observation count	284	53	12
Action count	17	7	2
Action type	Continues / Float	Continues / Float	Continues / Float
Trainer	PPO	PPO	PPO
Max step	5×10^6	27×10^6	12×10^6
Normalize	true	true	true
batch_size	4,096	2,048	4,096
buffer_size	49,600	8,192	49,600
beta	0.01	0.001	0.01
epsilon	0.2	0.2	0.2
gamma	0.9	0.9	0.9
hidden_unit	256	64	32
lambd	0.95	0.95	0.95
learning_rate	5×10^{-4}	5×10^{-4}	5×10^{-4}
num_epoch	3	3	3
num_layers	2	2	2

changes in the middle of the training, the timing learner increases its randomness to adapt to this wider search space.

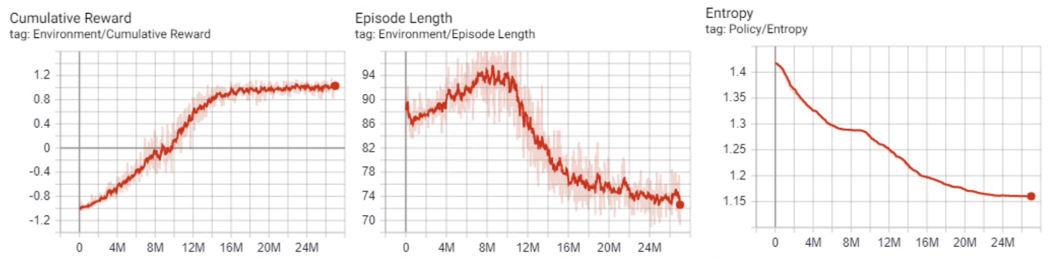


Figure 6.4: Tensorboard chart of the Reactive Learner training process.

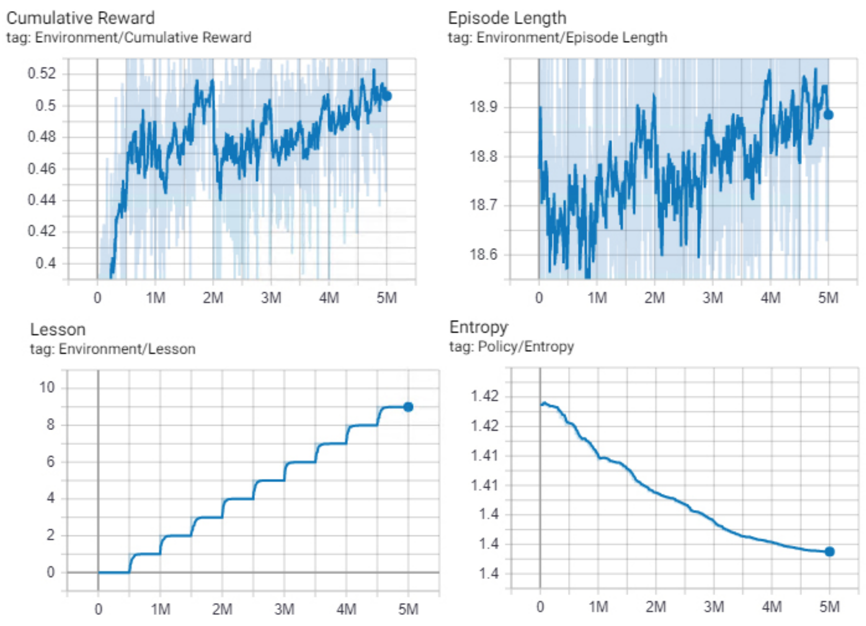


Figure 6.5: Tensorboard chart of the Proactive Learner training process.

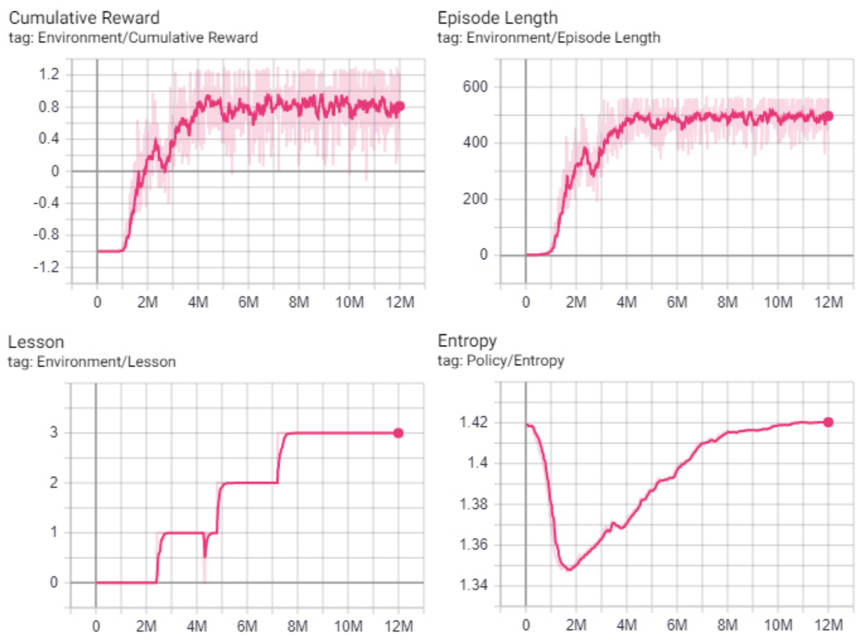


Figure 6.6: Tensorboard chart of the Timing Learner training process.

In the following subsections, we provide the results of our proposed framework by comparing them with other brains and models.

6.3 Proactive Brain Comparison: Train versus Heuristic with Skeleton Hand Model

In this experiment, the results of trained proactive brain are compared with the results of a hard coded heuristic brain. The heuristic brain gives a random value between [-1,1] as an action value and the target angles for each degree of freedom are calculated as described in Section 5.3. In these experiments, a skeleton hand model is used, to eliminate the positive effect of deformable model from the results. Two example comparisons for catching sphere and rectangular prism are displayed in Figure 6.7 and 6.8, respectively. The objects are thrown from the same initial position in both of the scenes. As far as we observe, there are two deficiencies of the heuristic brain compared to our trained brain. First of all, the finger movements are not fast enough to capture the object at the right time (cf. Figure 6.7). Besides, if the fingers move faster than they should, then the object cannot be caught with closed fingers. Moreover, the thumb movement, which is crucial for grasping, cannot be generated well with the heuristic brain (cf. Figure 6.8). As we observe from the results, our proposed proactive brain performs better than the heuristic brain for at generating finger movements for a successful catching.

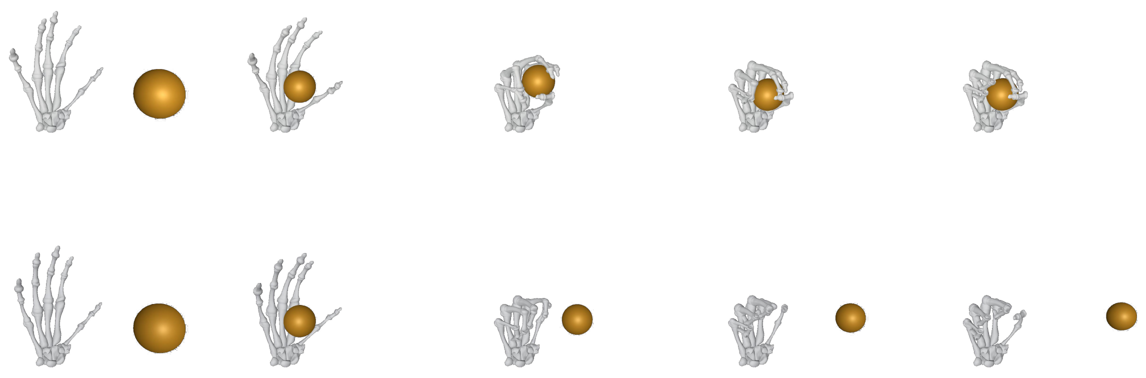


Figure 6.7: The comparison of trained proactive brain (up) and heuristic proactive brain (down) for catching a ball with skeleton hand model.

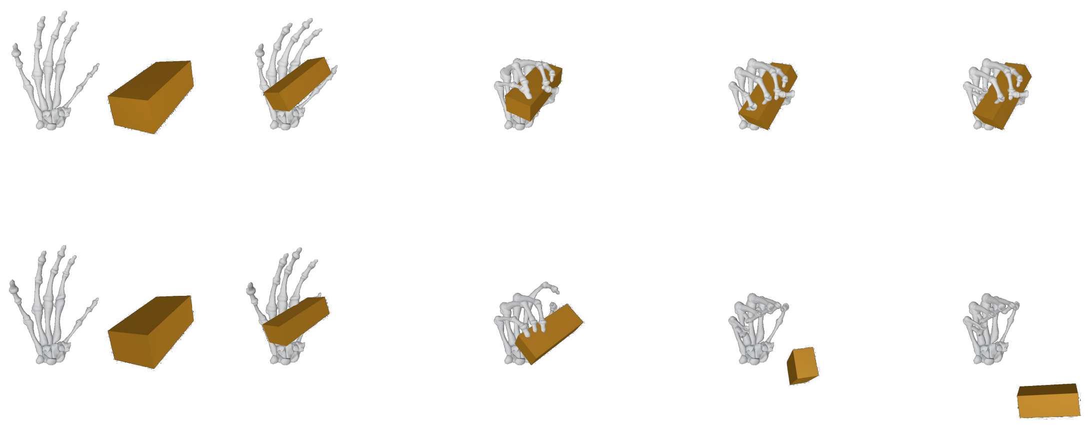


Figure 6.8: The comparison of trained proactive brain (up) and heuristic proactive brain (down) for catching a rectangular prism with skeleton hand model.

6.4 Proactive Brain Comparison: Trained versus Heuristic with Soft-body Hand Model

In this experiment, we again compare the results of the trained proactive brain with the heuristic brain described in the previous experiment, bu this time we use our proposed hand model with soft bodies. The purpose of this experiment is to examine if the positive effect of the soft body model resolves the deficiencies of the heuristic brain with the skeleton hand model. Figures 6.9 and 6.10 display two example comparisons for catching a sphere and a capsule. As expected, the capturing ability of the hand with heuristic brain increases as compared to the hand skeleton model. As seen in the figures, once contacted with the hand, the object is retained for a longer duration, but it slips after a time because of the wrong timing of the finger movements and the wrong movement of the thumb. Therefore, a more detailed model of the hand could not be sufficient for the heuristic brain to catch the objects. The results show that, the trained proactive brain is successful at catching the thrown objects.

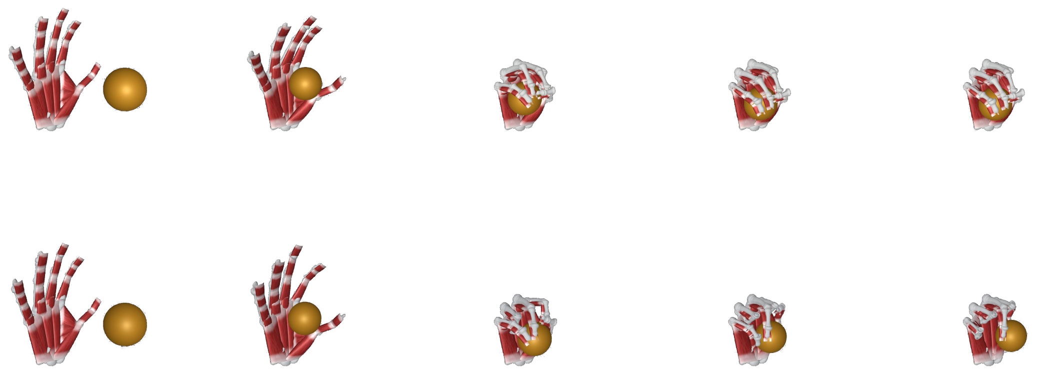


Figure 6.9: The comparison of trained proactive brain (up) and heuristic proactive brain (down) for catching a ball with the soft-body hand model.

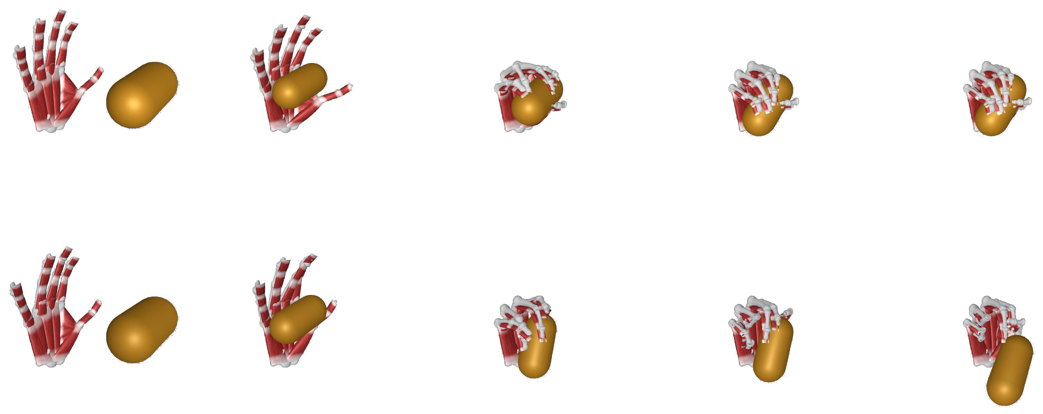


Figure 6.10: The comparison of trained proactive brain (up) and heuristic proactive brain (down) for catching a capsule with the soft-body hand model.

6.5 Hand Model Comparison: Skeleton versus Soft Body

In this experiment, we evaluate the effect of the used hand model on the success and naturalness of catching behaviors, by comparing our proposed hand model covered by soft bodies with a simple skeleton hand model consisting of only rigidbodies. A proactive brain is trained with curriculum learning for each of the hand model and used for generating the finger motions for both of the scenarios. Figures 6.11 and 6.12 display two example comparisons for catching a sphere and a capsule. In Figure 6.11, both hands are successful at capturing the sphere. But hand model with soft bodies are seen to achieve a more natural and stable catching behavior. In Figure 6.12, we see that the skeleton hand model is unsuccessful at catching the capsule, which is harder to catch than sphere because of its more complex geometry. Because of the deformable collision model we employed in the hand with soft bodies, a smoother collision occurs between the hand and the object, which gives time to the fingers for closing at the right time and capture the object before it bounces away from the hand. On the contrary, the object bounces off from the skeleton hand very fast because of the impact caused by the collision of the rigidbodies. The comparison of cumulative reward and episode length charts for both of the trainings are displayed in Figure 6.13. The figure depicts that as the training includes object with different geometries, the cumulative reward and episode length of the training with skeleton hand model decrease dramatically and cannot reach or exceed its maximum values again. This is due to the fact that, the skeleton hand shows a passable success for the objects with simpler geometry. However, when the object geometries get more complex, the hand skeleton mostly becomes unsuccessful at a stable catching behavior. This is compatible with the results of this comparison.

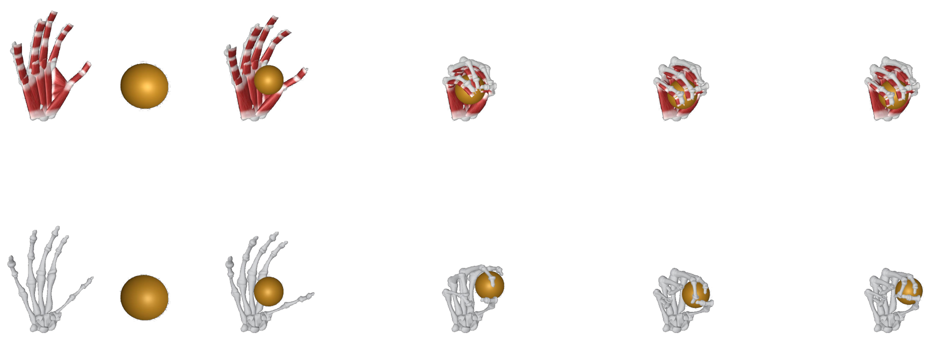


Figure 6.11: The comparison of the trained proactive brain for catching a ball with the soft-body hand model (up) and the skeleton hand model (down).

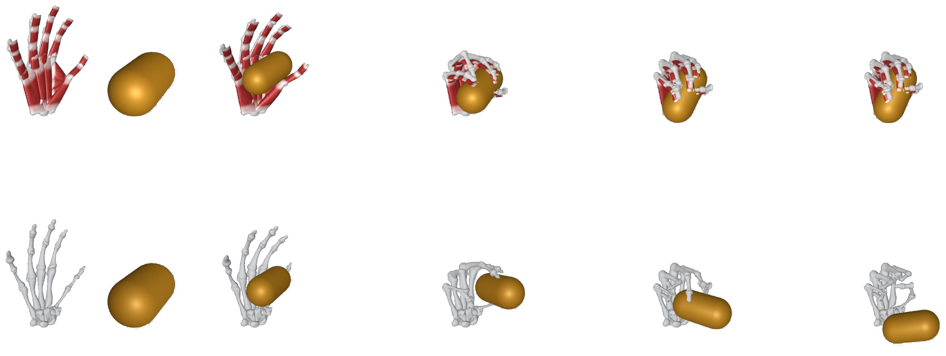


Figure 6.12: The comparison of the trained proactive brain for catching a capsule with the soft-body hand model (up) and the skeleton hand model (down).

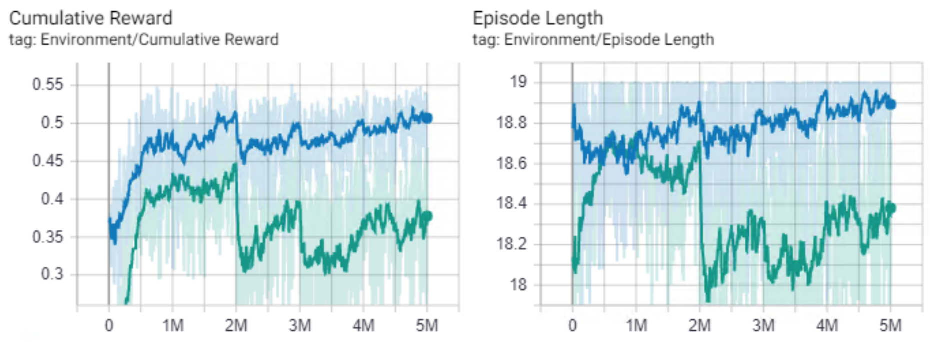
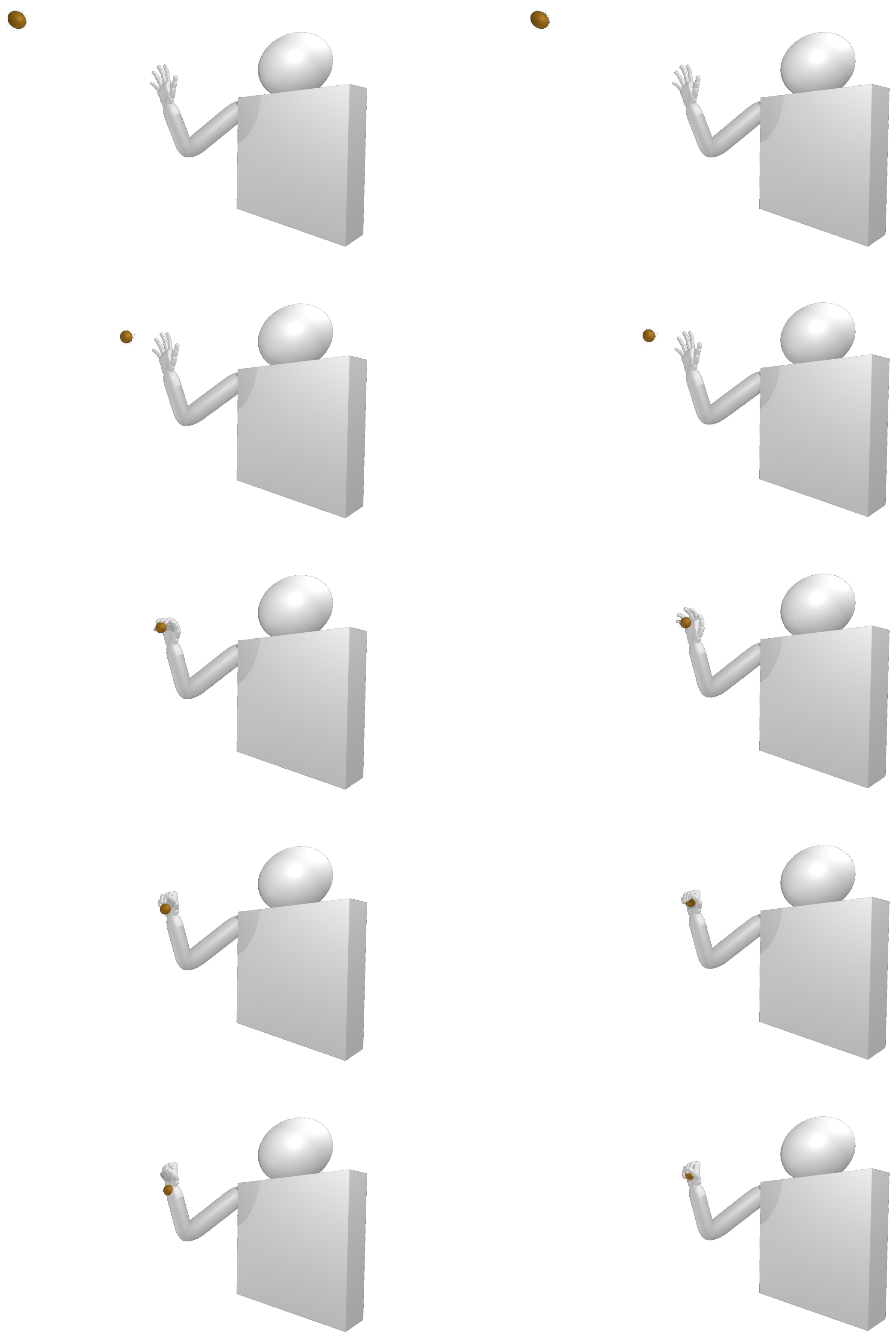


Figure 6.13: The comparison of the training charts of the proactive brain with the soft-body hand model (blue) and the skeleton hand model (cyan).

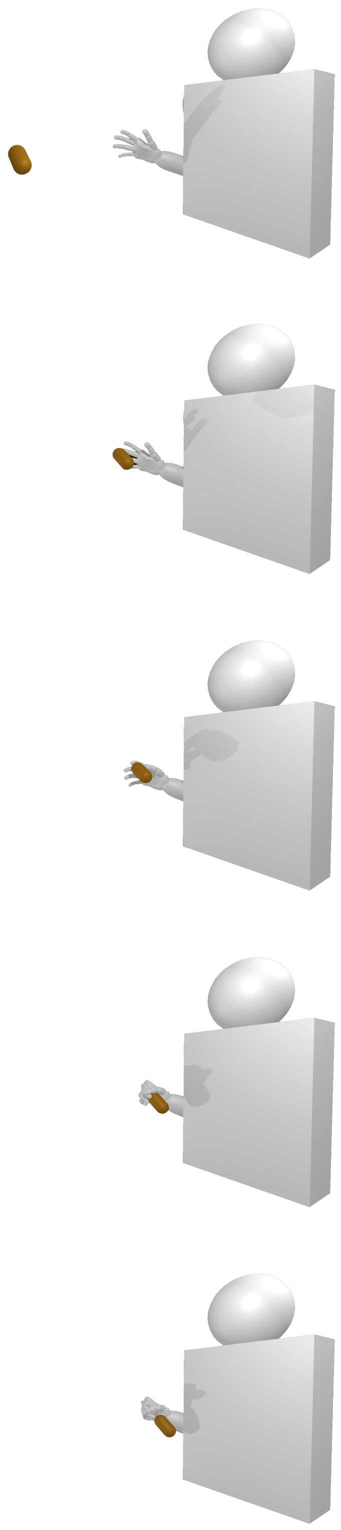
6.6 Meta Brain Comparison: Trained versus Heuristic

In this experiment, we compare the results of the trained meta brain with the results of a hard coded heuristic brain for arranging the start times of the reactive and proactive brains. In these experiments, we use the trained reactive and proactive brains and our proposed soft body hand model in both of the scenarios. In the heuristic approach, the reactive brain is run once the scene starts. When the object is 24 cm close to the hand, the proactive brain is run. In Figures 6.14 and 6.15, two example comparisons for catching a sphere and capsule are displayed. While the heuristic brain accomplishes catching most of the time, the trained meta brain cannot catch the object in some cases. Two of these cases are displayed in Figures 6.14 and 6.15, where two brains are compared for catching a sphere and a capsule, respectively. In these examples, it is seen that the proactive brain starts running earlier than it should; the object hits the closed fingers and falls down. In most of the cases, we observed that the start time of the reactive brain does not effect the success and the naturalness of the catching behavior.



(a) Heuristic meta brain (b) Trained meta brain

Figure 6.14: The comparison of the trained meta brain and the heuristic meta brain for catching a sphere with the soft-body hand model.


(a) Heuristic meta brain

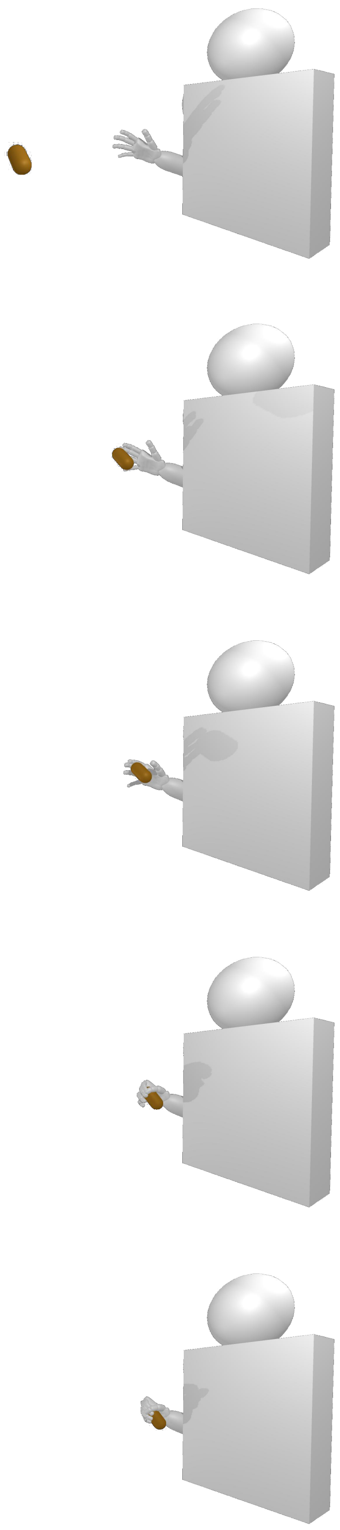

(b) Trained meta brain

Figure 6.15: The comparison of the trained meta brain and the heuristic meta brain for catching a capsule with the soft-body hand model.

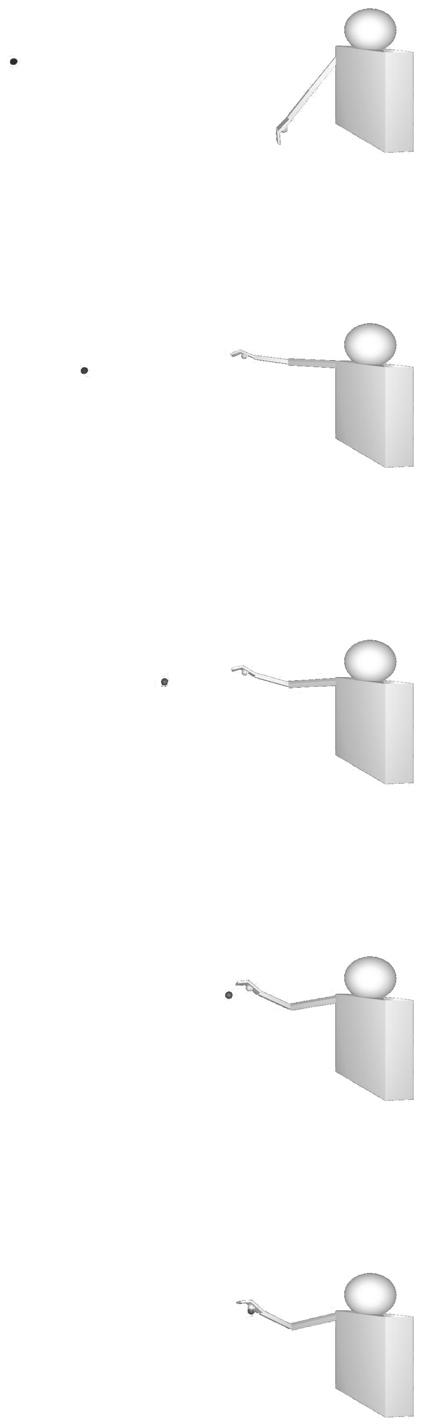
6.7 Reactive Brain Reward Comparison

In the following experiments, we trained the reactive brain by changing the weights of the step reward components explained in Section 5.2.

The total step reward value for the reactive learner is calculated with Equation 5.2, as a weighted sum of three components: *displacement penalty*, *route proximity reward* and *palm forward reward*, with the weights of -0.1, 0.35 and 0.65, respectively.

The first experiment is about the weights of the route proximity reward and palm forward reward. The route proximity reward is added to the total reward for guiding the arm to place the hand on the thrown objects accessible route. The palm forward reward helps to orient the hand such that the thrown object hits directly to the palm. When the weights of these two rewards are set to 0.5, the hand is placed on the route of the object as intended but the object hits the hand with nearly right angle. But for a successful catching, the velocity of the object should be aligned with the palm forward vector. When the weight of palm forward reward is increased to 0.65, the velocity vector gets aligned as intended. Figure 6.16 displays the results of these two weight combinations.

The second experiment is about the relation between the object-hand distance and the displacement penalty. The penalty for displacing the hand gets larger, as the object gets closer (cf. Equation 5.1). If we calculate the displacement penalty independent of the hand-object distance, then the arm continues to move even when the object is too close and as a result of this sudden movement the object and hand collides at a right angle, as seen in Figure 6.17. This causes difficulty for catching the object.


(a) Palm forward reward weight is 0.5

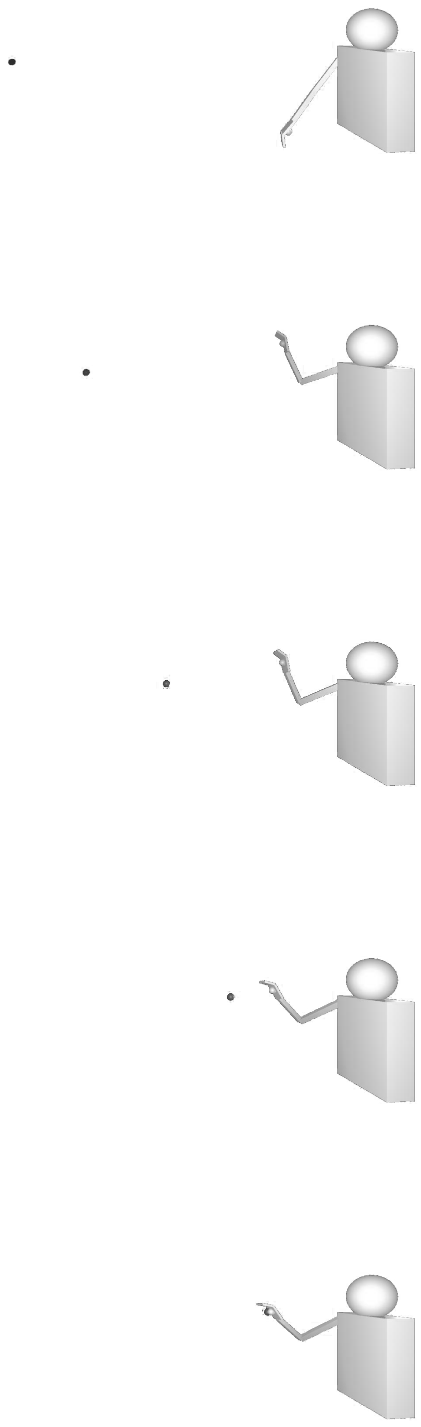
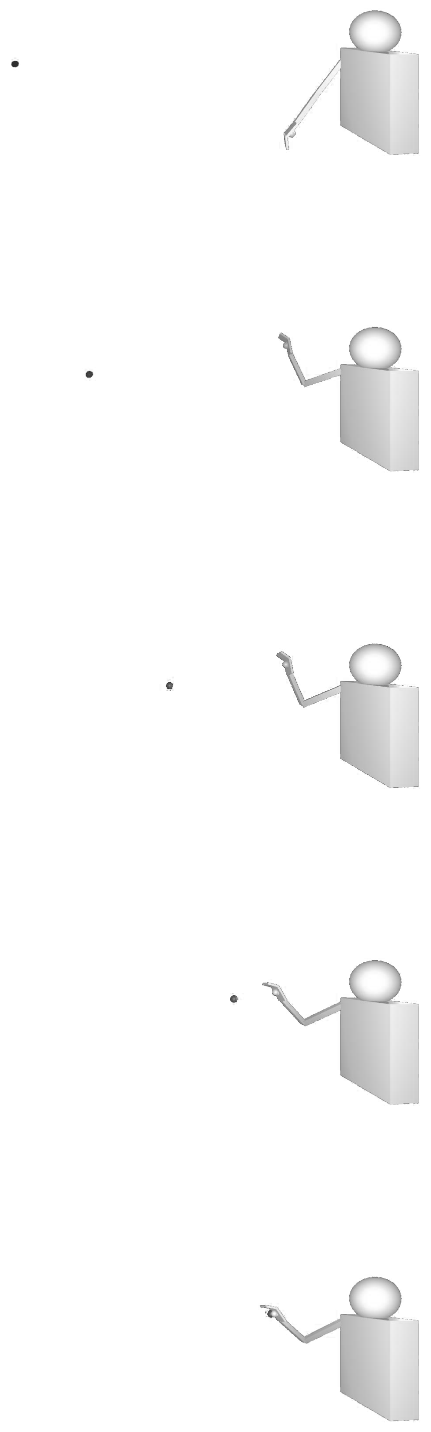

(b) Palm forward reward weight is 0.65

Figure 6.16: The effect of the higher weight on the palm forward reward.


(a) The displacement penalty is same for all training steps

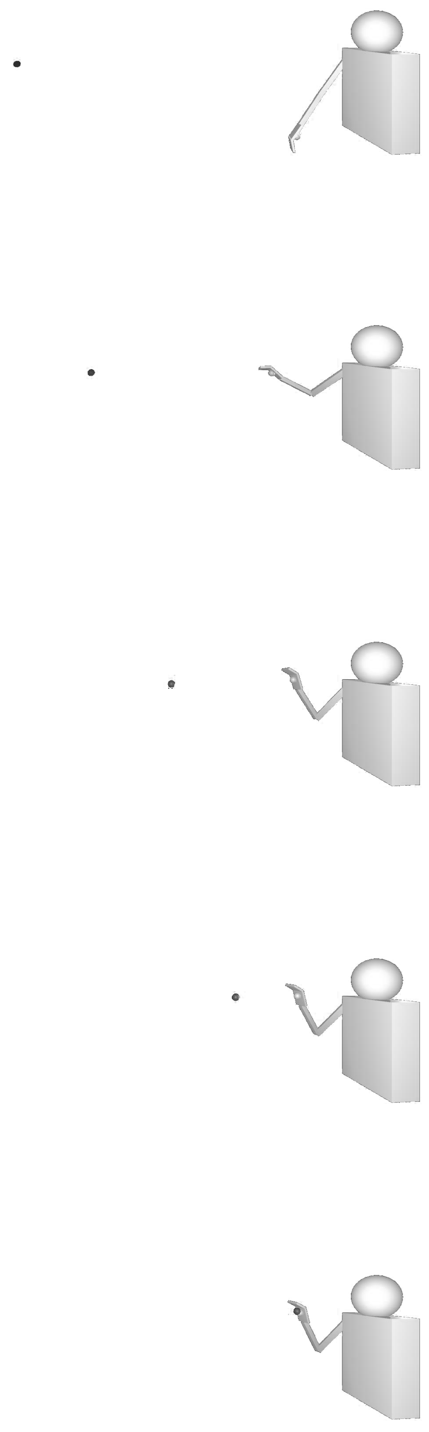

(b) The displacement penalty changes according to the hand-object distance

Figure 6.17: The effect of calculating the displacement penalty according to hand-object distance.

6.8 User Test Results

We conducted a user study to evaluate the naturalness of the generated catching motions. The user study consists of two sections for both the general catching animation and the detailed hand animation. A total of 25 people participated in the study. The 20 of the participants are men, and the 5 participants are women. Their ages are between 20 and 50. Generally, these people have experience in computer graphics. The videos are shown randomly to the users.

The main objective of this study is producing natural-looking ball catching animations. Because of that, animations of failed catching behaviours are not included in this user test.

Overall Animation Test

The first section of the user study compares the results of the meta brain trained by the timing learner with a hard coded heuristic brain, as explained in Section 6.6. With the heuristic brain, the timing of the reactive and proactive brains are procedurally determined. The reactive brain is run at the beginning of the scene and the proactive brain is run when the distance between the object and the hand is 24 cm. In both of the scenes, the initial states of the object and the environment are the same and the proactive and reactive brains trained with machine learning are used.

As shown in Figure 6.18 and Table 6.2, the results are generally close to each other. On the average, the timing learner is more successful, but the difference between them is minimal. There are several reasons why the difference is very small. First, the tremors occur when the arm animates kinematically and this effects the naturalness of the whole animation. Because of that, the scores can be similar. Second, all hand interactions are animated physically, which look very natural. Moreover, users gave average scores even to the uncaught objects, because we wanted them to rank the naturalness of the motion, not the catching abilities of the character.

In case 6, 9 and 10, the heuristic approach seems to score higher. In these scanarios, the object comes directly across the character, in which the heuristic brain produces actions similar to the trained brain.

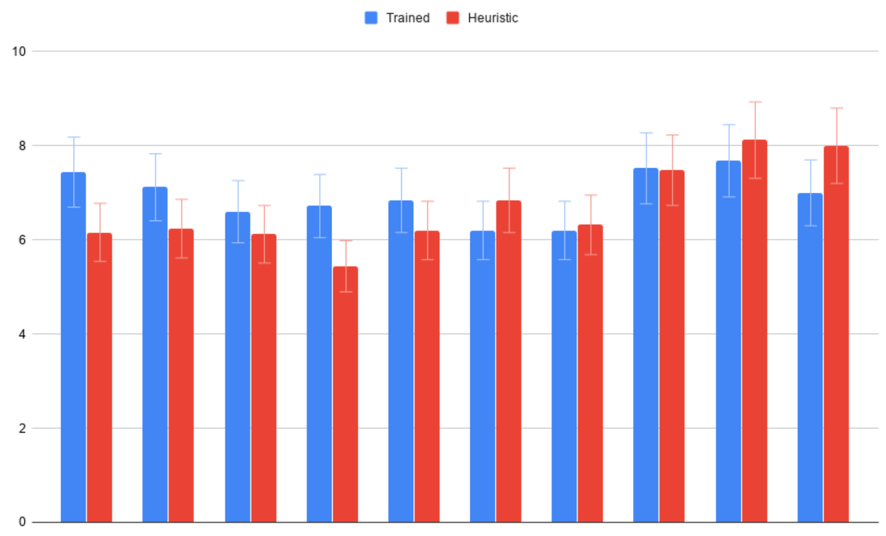


Figure 6.18: User test results for different meta brains.

Table 6.2: User test results for different meta brains.

Scene No	Train	Heuristic
1	7.44	6.16
2	7.12	6.24
3	6.60	6.12
4	6.72	5.44
5	6.84	6.20
6	6.20	6.84
7	6.20	6.32
8	7.52	7.48
9	7.68	8.12
10	7.00	8.00
Average	6.93	6.69

Detailed Hand Animation Test

In the second section of the user study, we requested the users to focus on and evaluate the hand animations. For this purpose, the videos begin with the starting of the proactive phase and show the hand in detail. The results of three different hand brain/model cases were included in the videos: the trained proactive brain with our proposed hand model with deformable bodies (Group 1), a hard-coded heuristic proactive brain with again our proposed hand model (Group 2) and the trained proactive brain with a hand model with only skeleton (Group 3). Three videos are captured for each scenario with the same initial object position, speed and animation state, but with a different hand brain/model case. In all of the scenarios, trained brains are used as the reactive and timing brains.

Table 6.3 and Figure 6.19 show that the trained proactive brain with our proposed hand model with deformable bodies gets more score than the others. In contrast to the first section of the user test, differences in scores are more noticeable. The trained brain with the skeleton hand model gets lower scores because the objects slip from the hand after catching. The heuristic agent with soft bodies gets relatively low score due to the failure to catch some objects and trembling of the joints in the positive and negative axes. The trained agent with soft bodies gets the highest scores for stable grasping.

Only for Scene 1, the heuristic brain with the soft body hand model (Group 2) gets a higher score than the trained brain with the soft body hand model (Group 1). In the scene with heuristic brain the fingers are trembling and in the scene with trained brain little and middle fingers overlap, since we ignore the collision of the fingers for physical stability concerns. With this result, we understand that users find overlapping fingers less natural than trembling.

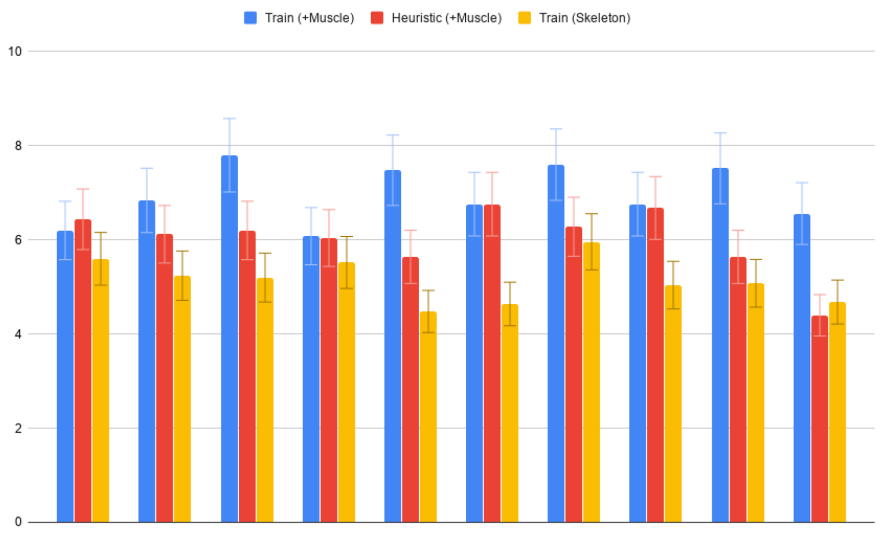


Figure 6.19: User test results for proactive phase

Table 6.3: User test results for the proactive phase.

Scene No	Train	Heuristic	Skeleton
1	6.20	6.44	5.60
2	6.84	6.12	5.24
3	7.80	6.20	5.20
4	6.08	6.04	5.52
5	7.48	5.64	4.48
6	6.76	6.76	4.64
7	7.60	6.28	5.96
8	6.76	6.68	5.04
9	7.52	5.64	5.08
10	6.56	4.40	4.68
Average	6.96	6.02	5.14

7 CONCLUSION

In this thesis, we present a framework to synthesize animations, where a thrown object is naturally captured with a physics-based hand model. We have contributed to the computer animation field by presenting a physics-based deformable hand model and a novel framework for creating catching motions composed of three parts, all trained with deep reinforcement learning, which are called reactive, proactive and timing learners.

A physical hand model was developed to catch the thrown objects more stably, with the two-layer hand model, where a soft body model is placed on the bone structure, thrown objects have been grasped easier, as shown in the result.

The hand must move to a suitable position in order for the thrown objects to be caught. For this purpose, the reactive phase brings the hand to the appropriate position with the rotation of the arm joints so that the hand can catch the object. The proper joint rotation angles of the arm are learned with deep Reinforcement Learning.

In the proactive phase, fingers are moved with physics-based animation to grasp the thrown object. To produce grasping animation, a behaviour model has been created with deep Reinforcement Learning. Since, the shape of the thrown objects are different. Training has been carried out using curriculum learning, which learns to catch simple objects first and increases the complexity of the training gradually.

The reactive and proactive phase should work in coordination with each other to get the hand in proper position and capture the thrown object. Timing learner is developed for deciding the start time of capture phases with the help of RL. This meta component controls the phases for the entire capture process and generates the entire capture animation.

The proposed study has some limitations. These limitations and intended future studies are given below.

7.1 Limitations

There are some limitations about the objects to be thrown. First of all, the object should be thrown across the character and to the right side. For an efficient reinforcement learning training, only the objects that can be captured should be thrown. For this reason, the throw directions were carefully selected during the training phase.

Secondly, the object is only rotated in the forward direction for better grasping, and does not have angular velocity. In this way, palm collides with the widest area of the object and an optimal grip animation is achieved.

Since the hand model has a physical structure, the working environment should be able to perform 1khz physics operations. Today's computers have 30-60 Hz integration steps for physics operations. At the beginning of the study, it was aimed to work at low frequencies. However, it was increased to 1 KHz due to the problems in physics calculations. Animations do not run in real-time due to the insufficient processing power in standard computers. Todays PCs that can run 1000 physics steps per second are not common.

Another limitation in the study is about the finger bones and joints. Under normal conditions, bones should collide with each other. However, if the collision function is activated for bones connected to the same joint, they start to push each other. For this reason, collision control is not activated for the connected fingers. Collision check is performed only for the last part of the fingers. Therefore, it is rarely observed that one finger overlaps to the other.

Some other limitations occur due to the kinematic structure of arm motion. According to general catching motion, when the object collides with the hand, it needs to go back a little because of collision impact. However, the hand's position does not change because the wrist to which hand is attached is animated kinematically. This situation negatively affects the naturalness of the general movement. Moreover the motions of the arm is not too smooth. Although the main focus of the study is the hand's physical animation, these limitations because of arm motion effect the results negatively.

7.2 Future Work

We model the joints on the fingers are modeled with certain rotational limits. However, human fingers also have some important relations between each other that restrict their movements. For example, the ring finger must turn 45 degrees for the little one to turn 90 degrees. Implementing these constraints would increase the natural appearance of the movement. Besides, reinforcement learning training would be shorter as the search space decreases.

The muscle structure of the proposed hand model, which is developed to provide soft contacts is complex and heavy. There are nearly 1400 mass points in our hand model and these mass points interact with the thrown object. Because of that, the physics operations can not run in realtime. Developing a more optimized deformable model for hand is an important future direction.

The thrown object is selected as cube, sphere, capsule and prism. These objects have regular mesh geometry. Because of that, to catch a regular object is easier than an irregular object. For the next step, irregular objects are also added to the system to extend its capability.

As mentioned in the limitations section, the motion of the arm is performed kinematically. This kinematic motion is preferred because it speeds up machine learning training. If the arm is physically modeled, the overall animation of the capture would look more natural.

The reactive and proactive phases are running as a separate operation. They are trained in their environment. Arm and hand can be run simultaneously in the proactive phase for increase naturalness of the catching animation. In the proactive phase, the arm can rotate to hand to produce more stable and natural-looking animation.

In our study, only the right arm and hand are taken into account. With the same framework, the left arm and hand can also be trained. If two limbs are coordinated for capture, the area where the hand can catch the object would be much larger. With a meta brain that coordinates the two limbs, the capture animation of the objects can be synthesized more naturally.

In the developed system, the position of the shoulder is fixed, and the arm moves accordingly. For this reason, it becomes difficult for the hand to reach the proper position Intelligent some cases. If the ability of rotation is given to the pelvis, it would be much easier for the hand to reach the proper position. Thus, capturing the thrown objects would be more natural and realistic, and fewer restrictions would be applied.

REFERENCES

[1] Yeo, S.H., Lesmana, M., Neog, D.R., Pai, D.K., Eyecatch: Simulating visuomotor coordination for object interception, *ACM Trans Graph*, 31(4), **2012**. URL `https://doi.org/10.1145/2185520.2185538`.

[2] Eom, H., Han, D., Shin, J.S., Noh, J., Model predictive control with a visuomotor system for physics-based character animation, *ACM Transactions on Graphics*, 39(1), 1–11, **2020**. URL `https://doi.org/10.1145/3360905`.

[3] Hirt, B., *Hand and Wrist Anatomy and Biomechanics: A Comprehensive Guide*, Thieme, Stuttgart, Germany New York, **2017**.

[4] Schunke, M., *Thieme Atlas of Atatomy*, Thieme Medical Publishers, Inc, New York, New York, **2014**.

[5] ml-agents/background-machine-learning.md at master, unity-technologies/ml-agents, github. URL `https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Background-Machine-Learning.md`, (Accessed on 07/16/2020).

[6] Introducing ml-agents toolkit v0.2: Curriculum learning, new environments, and more - unity technologies blog. URL `https://blogs.unity3d.com/2017/12/08/introducing-ml-agents-v0-2-curriculum-learning-new-environments-and-more/`, (Accessed on 07/16/2020).

[7] ml-agents/ml-agents-overview.md at master, unity-technologies/ml-agents, github. URL `https://github.com/Unity-Technologies/ml-agents/blob/master/docs/ML-Agents-Overview.md`, (Accessed on 07/16/2020).

[8] Pid controller - wikipedia. URL `https://en.wikipedia.org/wiki/PID\_controller`, (Accessed on 07/17/2020).

[9] Complete anatomy | image courtesy of complete anatomy. URL `https://3d4medical.com`, (Accessed on 08/05/2020).

[10] Kaelbling, L.P., Littman, M.L., Moore, A.W., Reinforcement learning: A survey, *Journal of Artificial Intelligence Research*, 4, 237–285, **1996**. URL https://doi.org/10.1613/jair.301.

[11] Proximal policy optimization. URL https://openai.com/blog/openai-baselines-ppo/, (Accessed on 08/18/2020).

[12] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., Proximal policy optimization algorithms, *arXiv preprint arXiv:170706347*, **2017**.

[13] Bengio, Y., Louradour, J., Collobert, R., Weston, J., Curriculum learning, *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, ACM Press, **2009**. URL https://doi.org/10.1145/1553374.1553380.

[14] Hengst, B., *Hierarchical Reinforcement Learning*, Springer US, Boston, MA, 495–502, **2010**. URL https://doi.org/10.1007/978-0-387-30164-8.

[15] Dayan, P., Hinton, G.E., Feudal reinforcement learning, *Advances in Neural Information Processing Systems*, **1993**, volume 5, 271–278.

[16] Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M., Lange, D., Unity: A general platform for intelligent agents, *CoRR*, abs/1809.02627, **2018**. URL http://arxiv.org/abs/1809.02627.

[17] ml-agents/training-configuration-file.md at master, unity-technologies/ml-agents, github. URL https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-Configuration-File.md, (Accessed on 07/19/2020).

[18] Müller, M., Heidelberger, B., Teschner, M., Gross, M., Meshless deformations based on shape matching, *Proceedings of ACM SIGGRAPH'05*, ACM Press, **2005**. URL https://doi.org/10.1145/1186822.1073216.

[19] Geijtenbeek, T., Pronost, N., Interactive character animation using simulated physics: A state-of-the-art review, *Computer Graphics Forum*, 31(8), 2492–2515, **2012**. URL https://doi.org/10.1111/j.1467-8659.2012.03189.x.

[20] Max payne 3. URL https://www.rockstargames.com/maxpayne3/, (Accessed on 07/27/2020).

[21] Grand theft auto v. URL https://www.rockstargames.com/V/, (Accessed on 07/27/2020).

[22] Introducing learned motion matching - ubisoft montreal. URL https://montreal.ubisoft.com/en/introducing-learned-motion-matching/, (Accessed on 08/25/2020).

[23] Holden, D., Kanoun, O., Perepichka, M., Popa, T., Learned motion matching, *ACM Transactions on Graphics*, 39(4), **2020**. URL https://doi.org/10.1145/3386569.3392440.

[24] Unbehauen, H., *Control systems, robotics and automation*, Eolss Publishers Co. Ltd, Oxford, **2009**.

[25] Neff, M., Kipp, M., Albrecht, I., Seidel, H.P., Gesture modeling and animation based on a probabilistic recreation of speaker style, *ACM Transactions on Graphics*, 27(5), 1–24, **2008**.

[26] Ip, H., Chan, S., Lam, M., Hand gesture animation from static postures using an anatomy-based model, *Proceedings of the Computer Graphics International (CGI'2000)*, **2000**, 29–36.

[27] Shankar, V., Ghosh, D., Dynamic hand gesture synthesis and animation using image morphing technique, *Proceedings of the IET International Conference on Visual Information Engineering*, **2006**, 543–548.

[28] Aleotti, J., Caselli, S., Grasp programming by demonstration in virtual reality with automatic environment reconstruction, *Virtual Reality*, 16(2), 87–104, **2010**. URL https://doi.org/10.1007/s10055-010-0172-8.

[29] Pollard, N.S., Zordan, V.B., Physically based grasping control from example, *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation - SCA '05*, ACM Press, **2005**. URL https://doi.org/10.1145/1073368.1073413.

[30] Li, Y., Fu, J.L., Pollard, N.S., Data-driven grasp synthesis using shape matching and task-based pruning, *IEEE Transactions on Visualization and Computer Graphics*, 13(4), 732–747, **2007**. URL `https://doi.org/10.1109/tvcg.2007.1033`.

[31] Mordatch, I., Popovic, Z., Todorov, E., Contact-invariant optimization for hand manipulation, *in Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, (SCA '12), 137–144*, The Eurographics Association, **2012**. URL `http://diglib.eg.org/handle/10.2312/SCA.SCA12.137-144`.

[32] Tian, H., Wang, C., Manocha, D., Zhang, X., Realtime hand-object interaction using learned grasp space for virtual environments, *IEEE Transactions on Visualization and Computer Graphics*, 25(8), 2623–2635, **2019**. URL `https://doi.org/10.1109/tvcg.2018.2849381`.

[33] Chi, D., Costa, M., Zhao, L., Badler, N., The EMOTE model for effort and shape, *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '00*, ACM Press, **2000**. URL `https://doi.org/10.1145/344779.352172`.

[34] Hartmann, B., Mancini, M., Pelachaud, C., Formational parameters and adaptive prototype instantiation for MPEG-4 compliant gesture synthesis, *Proceedings of Computer Animation 2002 (CA 02)*, IEEE Comput. Soc. URL `https://doi.org/10.1109/ca.2002.1017516`.

[35] Fernández-Baena, A., Montaño, R., Antonijoan, M., Roversi, A., Miralles, D., Alías, F., Gesture synthesis adapted to speech emphasis, *Speech Communication*, 57, 331–350, **2014**. URL `https://doi.org/10.1016/j.specom.2013.06.005`.

[36] Shen, X., Hua, G., Williams, L., Wu, Y., Dynamic hand gesture recognition: An exemplar-based approach from motion divergence fields, *Image and Vision Computing*, 30(3), 227–235, **2012**. URL `https://doi.org/10.1016/j.imavis.2011.11.003`.

[37] Bao, J., Song, A., Guo, Y., Tang, H., Dynamic hand gesture recognition based on SURF tracking, *Proceedings of the International Conference on Electric Information and Control Engineering*, IEEE, **2011**. URL `https://doi.org/10.1109/iceice.2011.5777598`.

[38] Kılıboz, N., Gudukbay, U., A hand gesture recognition technique for human–computer interaction, *Journal of Visual Communication and Image Representation*, 28, 97–104, **2015**. URL `https://doi.org/10.1016/j.jvcir.2015.01.015`.

[39] Wheatland, N., Wang, Y., Song, H., Neff, M., Zordan, V., Jörg, S., State of the art in hand and finger modeling and animation, *Computer Graphics Forum*, 34(2), 735–760, **2015**. URL `https://doi.org/10.1111/cgf.12595`.

[40] Kry, P.G., Pai, D.K., Interaction capture and synthesis, *ACM SIGGRAPH '05 Sketches*, ACM Press, **2005**. URL `https://doi.org/10.1145/1187112.1187262`.

[41] Zhao, W., Zhang, J., Min, J., Chai, J., Robust realtime physics-based motion control for human grasping, *ACM Trans Graph*, 32(6), **2013**. URL `https://doi.org/10.1145/2508363.2508412`.

[42] Liu, C.K., Synthesis of interactive hand manipulation, in *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, 163–171, **2008**. URL `http://diglib.eg.org/handle/10.2312/SCA.SCA08.163-171`.

[43] Jain, S., Liu, C.K., Controlling physics-based characters using soft contacts, *Proceedings of the 2011 SIGGRAPH Asia Conference*, Association for Computing Machinery, New York, NY, USA, **2011**, SA '11. URL `https://doi.org/10.1145/2024156.2024197`.

[44] Talvas, A., Marchal, M., Duriez, C., Otaduy, M.A., Aggregate constraints for virtual manipulation with soft fingers, *IEEE Transactions on Visualization and Computer Graphics*, 21(4), 452–461, **2015**. URL `https://doi.org/10.1109/tvcg.2015.2391863`.

[45] Andrews, S., Kry, P., Goal directed multi-finger manipulation: Control policies and analysis, *Computers & Graphics*, 37(7), 830–839, **2013**. URL `https://doi.org/10.1016/j.cag.2013.04.007`.

[46] Çimen, G., Kavafoğlu, Z., Kavafoğlu, E., Çapın, T., Gürcay, H., Skill learning based catching motion control, *Computer Animation and Virtual Worlds*, 26(3-4), 217–225, **2015**. URL `https://doi.org/10.1002/cav.1659`.

[47] Jörg, S., *Data-Driven Hand Animation Synthesis*, 1–13, **2016**.

[48] Irimia, A.S., Chan, J., Mistry, K., Wei, W., Ho, E., Emotion transfer for hand animation, *in Proceedings of the Motion, Interaction and Games, MIG '19*, **2019**, 1–2.

[49] Zhao, W., Chai, J., Xu, Y.Q., Combining marker-based mocap and rgb-d camera for acquiring high-fidelity hand motion data, *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Goslar, DEU, **2012**, SCA '12, 33–42.

[50] Rijpkema, H., Girard, M., Computer animation of knowledge-based human grasping, *ACM Computer Graphics (Proc SIGGRAPH '91)*, 25(4), 339–348, **1991**. URL `https://doi.org/10.1145/127719.122754`.

[51] Kurihara, T., Miyata, N., Modeling deformable human hands from medical images, *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Goslar, DEU, **2004**, SCA '04, 355–363. URL `https://doi.org/10.1145/1028523.1028571`.

[52] Albrecht, I., Haber, J., Seidel, H.P., Construction and animation of anatomically based human hand models, *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Goslar, DEU, **2003**, SCA '03, 98–109.

[53] Sueda, S., Kaufman, A., Pai, D.K., Musculotendon simulation for hand animation, *ACM SIGGRAPH 2008 Papers*, Association for Computing Machinery, New York, NY, USA, **2008**, SIGGRAPH '08. URL `https://doi.org/10.1145/1399504.1360682`.

[54] Sifakis, E., Neverov, I., Fedkiw, R., Automatic determination of facial muscle activations from sparse motion capture marker data, *ACM Trans Graph*, 24(3), 417–425, **2005**. URL `https://doi.org/10.1145/1073204.1073208`.

[55] Barrielle, V., Stoiber, N., Cagniart, C., Blendforces: A dynamic framework for facial animation, *Proceedings of the 37th Annual Conference of the European Association for Computer Graphics*, Eurographics Association, Goslar, DEU, **2016**, EG '16, 341–352.

[56] Lee, S.H., Sifakis, E., Terzopoulos, D., Comprehensive biomechanical modeling and simulation of the upper body, *ACM Trans Graph*, 28(4), **2009**. URL `https://doi.org/10.1145/1559755.1559756`.

[57] Unity real-time development platform | 3d, 2d vr & ar visualizations. URL `https://unity.com/`, (Accessed on 07/20/2020).

[58] Github - nvidiagameworks/physx: Nvidia physx sdk. URL `https://github.com/NVIDIAGameWorks/PhysX`, (Accessed on 07/20/2020).

[59] Github - unity-technologies/ml-agents: Unity machine learning agents toolkit. URL `https://github.com/Unity-Technologies/ml-agents`, (Accessed on 07/20/2020).