Can We Teach Computers Magic Tricks?

Yunhao Luo
Master of Computer Science

School of Computer Science
McGill University
Montreal, Quebec, Canada

July 23, 2024

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Computer Science

Contents

List of Figures

List of Tables

Abstract

This thesis explores the challenges of synthesizing animations of a human hand interacting with various objects and aims to answer the following question: Can we teach computers magic tricks? To address this question, we experiment with various methods and present two systems for synthesizing animations of hand-object interactions. The first system simulates a fully articulated human hand and arm catching and throwing objects in a rigid body simulation. We design a nominal controller based on a finite state machine to control the hand and use reinforcement learning to automatically tune the control parameters. The second system simulates a soft hand manipulating thin-shell cards to change their configuration in a soft body simulation. In this system, we use optimization to find the movement trajectory for each fingertip. As a result, we successfully train reinforcement agents to complete various catching and throwing tasks using a nominal controller, and we optimize multiple card sleights performed by fingertips simulated as soft bodies. Hence, we believe that, with these two systems as the foundation, the ultimate goal of teaching computers to perform magic tricks in simulation is within reach.

Abrégé

Cette thèse explore les défis liés à la synthèse d'animations d'une main humaine interagissant avec divers objets et vise à répondre à la question suivante : peut-on apprendre aux ordinateurs à faire des tours de magie? Pour répondre à cette question, nous expérimentons diverses méthodes et présentons deux systèmes pour la synthèse d'animations d'interactions main-objet. Le premier système simule une main et un bras humains entièrement articulés, attrapant et lançant des objets dans une simulation de corps rigides. Nous concevons un contrôleur nominal basé sur une machine à états finis pour contrôler la main et utilisons l'apprentissage par renforcement pour régler automatiquement les paramètres de contrôle. Le deuxième système simule des bouts de doigts manipulant des cartes à enveloppe mince pour modifier leur configuration dans une simulation de corps mou. Dans ce système, nous utilisons un processus d'optimisation pour trouver la trajectoire de mouvement de chaque bout de doigt. En résumé, nous réussissons à entraîner des agents de renforcement à accomplir diverses tâches d'attraper et de lancer en utilisant un contrôleur nominal, et nous optimisons plusieurs tours de cartes réalisés par des bouts de doigts simulés en tant que corps mous. Par conséquent, nous croyons qu'avec ces deux systèmes comme fondement, l'objectif ultime d'apprendre aux ordinateurs à exécuter des tours de magie en simulation est à portée de main.

Contribution

The thesis writing are supervised and advised by Professor Paul Kry at the McGill University and Professor Sheldon Andrews at the École de technologie supérieure.

The first project of catching and throwing mentioned in Section 3.1 and Section 4.1 are adapted from Luo et al. (2021), which is an original work of mine in collaboration with the fellow student Kaixiang Xie at McGill University. This project is supervised and advised by Professor Paul Kry and Professor Sheldon Andrews. I am responsible for designing and implementing the controller and simulation framework, modeling the hand and arm, applying reinforcement learning to the controller, conducting major experiments, and writing the paper. The fellow student Kaixiang helps me in designing and training the reinforcement learning agent as well as conducting the experiments relating to reinforcement learning.

The second project of optimizing card manipulation mentioned in Section 3.2 and Section 4.2 is an original work of mine supervised by Professor Paul Kry and Professor Sheldon Andrews, where I implemented all code and conducted all experiments.

Acknowledgements

All project are supervised and advised by Professor Paul Kry at the McGill University and Professor Sheldon Andrews at the École de technologie supérieure.

Especially, the project of catching and throwing mentioned in Section 3.1 and Section 4.1 was funded in part by NSERC Discovery 2018-05665, and in collaboration with the fellow student Kaixiang Xie at McGill University.

1

Introduction

Synthesizing realistic hand animation has always been an interesting but challenging task in the field of computer animation. To synthesis convincing hand animation, we would like to teach computer how to do hand manipulation by itself in physical simulation. Amongst the many hand motions, we choose to synthesis catching, throwing and in-hand card manipulations. These cases are chosen for their complex motion and rich involvement of fingers. Hence, if a computer can master them all, it is reasonable to say they can master more general hand manipulation tasks.

Various methods have been proposed in the Computer Graphics community for modeling and animation virtual human hands in the past decades. Related works have also been done on physical anthropomorphic hands in the Robotics community. Amongst the many different methods, including but not limited to motion capturing, data-driven synthesis, physics-based algorithms, and surface modeling. But none of them have con-

structed a system for synthesizing animations of fully articulated hand model interacting with thin shells in real time. Therefore, we aim to fill this gap by synthesizing animation of human hands doing complex tasks.

First, we simulate a fully articulated human hand with an arm for catching and throwing objects in a rigid body simulation. We design a nominal controller for animating a fully-articulated physics-based human arm model including the hands and fingers, where all joints are preserved and animated, to catch and throw objects. Here, the term nominal controller refers to a minimal functional finite state machine controller, which should be assisted by reinforcement learning to achieve its full potential. The controller is based on a finite state machine that defines the target poses for proportional-derivative (PD) control of the hand, as well as the orientation and position of the center of the palm using the solution of an inverse kinematics solver on the palm and the arm. Especially, the PD targets of the fingers are generated using a mixture of base hand poses. We then use reinforcement learning (RL) to train agents to improve the robustness of the nominal controller for achieving many different goals. Imitation learning based on trajectories output by a numerical optimization is used to accelerate the training process. The success of our controllers is demonstrated by a variety of throwing and catching tasks, including flipping objects, hitting targets, and throwing objects to a desired height, and for several different objects, such as cans, spheres, and rods.

Creating convincing animations of human catching and throwing is important for many computer animation, virtual reality, and video game applications. While the most straight forward method is to use motion capture, this approach may require capturing many individual motion clips, or blending between multiple clips so that many different scenarios can be handled. Another strategy is to synthesize hand and arm motions using procedural and learning-based approaches. Catching and throwing control has been previously investigated in this context, but most approaches use a simplified hand model that is not fully articulated. Rather, our approach uses a fully articulated arm and

hand model to generate physically plausible and human-like animations for throwing and catching. At the core of our method is a nominal controller that can catch and throw objects of different shapes, and for a variety of different tasks. With the help of reinforcement learning, controller parameters are learned such that the controller may be used for many different simulation states.

Inspired by the work of Pollard and Zordan (2005), the controller is structured as a finite state machine (FSM) that sets the desired hand pose and palm position and orientation for several phases: approaching, pre-grasp, stable grasp, and release. This is likewise similar to the work of Andrews and Kry (2013), but with a focus on catching and throwing rather than in-hand manipulation. Each state of the FSM defines a Hermite curve specifying the trajectory of the target position for the hand to track, as well as the target rotation. Given the target position and rotation, inverse kinematics (IK) is used to solve for the corresponding joint angles. Most of the control points for Hermite curves are computed by the planning algorithm, except those of the `THROW` state, which are controlled by the reinforcement learning agent. The main idea is that it is relatively straightforward to set up the planner to produce a trajectory resembling what is necessary to accomplish the task, and the tacit knowledge necessary for successful completion of the task can be acquired through reinforcement learning.

Our goal is to create a controller that can be easily applied to human hand models to generate natural catching and throwing animations by focusing on three objectives:

- *Simple parameter tuning*. The user should only need to adjust a small number of control parameters, and the remaining ones are automatically determined.

- *Natural motion*. The catching and throwing motions generated by our system should be convincing, with smooth and continuous trajectories, no abrupt unnatural movements, and without awkward poses and joint rotations that are not humanly possible.

- *Robustness*. The controllers should be able to handle several different types of objects, and in this work, we work with spheres, cylinders, cuboids, and clubs; the shape of these objects can vary within a certain tolerance.

Second, we use optimization methods to make human hand doing more complex magic tricks in a soft body simulation. For the card magic tricks, we referred to the classic book by Hugard and Braue (1974).

Card magic is the art of deceiving. In a magic show, each individual unit performance is considered a trick, and in the course of a card trick, sleights of hands are used as secret processes to manipulate the card in a deceiving way. When employing the sleights, techniques are used to perfect the details. Meanwhile, the flourishes are used to garnish the performance. In this thesis we mainly explore two sleights: Card Snapping, which means snapping cards apart evenly, and Double Lift, which means lifting 2 cards as one. A nomenclature of card tricks can be found in the Appendix.

Cards are hard to simulate due to their relatively small dimensions and rich deformation behaviour. The cards used in card manipulation are called playing cards, which includes a variety of different card formats. Amongst them, we choose to simulate the poker card, which is the most-used format in modern card magics. A standard poker card typically has dimensions of 88.9 mm long by 63.5 mm wide. The thickness of poker cards are not specified in a universal standard and can approximately range from 0.14 mm to 0.36 mm. Hence, we choose the median value 0.25 mm as the default card thickness in our simulation. The playing cards are thin and delicate comparing to the dimensions of human hands. To preserve their fidelity in simulation, we use deformable thin shells with thickness to model these poker cards.

Human hands have complex biological structures, and cards are relatively thin comparing to the hand. Therefore, careful modeling is critical to the success. We model the fingertips of the hand as deformable 3D body using tetrahedral mesh and the card as deformable thin shell using triangle mesh. To ensure the hand and card are penetration

free, we use the codimensional incremental potential contact (Codim-IPC) model by Li et al. (2021). Additionally, to simulate such delicate scenarios, careful parameter tuning and model discretization are important. We explore a wide range of simulation parameters and different discretization of both hand and card. Besides, the initial conditions of the card and hand are important to the stability and efficiency of simulation. Hence, we warm start the simulation by saving a suitable initial state to speedup the simulation and avoid numerical turbulence.

To control the hand model with such a high degrees of freedom, we use optimization method with bone based control. Full simulation of joints and muscles has been done before by Lee et al. (2019); they use two-level imitation learning algorithm that handles a full-body musculoskeletal model with 346 muscles. But a full simulation is too time consuming for our uses. Instead, we control part of the tetrahedral model to drive the entire hand. This is analog to how human bones rotating around the joints and driving the surrounding tissues. The center part of the mesh are driven by Dirichlet boundary condition to simulate the mechanisms of human joints and muscles. For the optimization, considering the expensive time and space cost of soft body simulation, we use Multiobjective Tree-structured Parzen Estimator (MOTPE) by Ozaki et al. (2020), which is designed to address expensive multi-objective optimization problems.

Overall, we build two different systems to simulate different complex hand manipulation jobs. One is a nominal controller assisted by reinforcement learning to complete complex catching and throwing, and the other is an optimization framework of synthesizing in-hand card manipulation. They both provide novel ways to approach synthesizing animation of human hand doing complex jobs.

2

Literature

A complete physics-based system that synthesizes convincing hand animation can generally be decomposed into several components: hand modeling, simulation environment, and control algorithm. Multiple related surveys already exist. Wheatland et al. (2015) presented a survey on virtual hand and finger modeling and animation. Zhang et al. (2020) presented a survey on robotic hands. Kwiatkowski et al. (2022) presented a survey on the reinforcement learning methods in character animation. Therefore, this report focus more on the more recent technological advancements in hand control and animation while still accounting previous works.

2.1 Modeling and Simulation

Hand modeling is the foundation of hand control and animation. Under physical simulation, proper joint constraints and shaping of the hand components can be critical to

generating realistic hand motion sequences. Various hand models were presented in the Graphics community, which can be found in the aforementioned survey by Wheatland et al. (2015). Meanwhile, different types of robotics hand were created for fitting piratical purpose in the Robotics community. Abondance et al. (2020) created 4 finger deformable robotic hand that has strong grasping capabilities. Xiong et al. (2016) designed and implemented an anthropomorphic hand that can replicate human grasping functions.

Physical simulation techniques are critical in creating a fast and reliable environment for hand animation. A single interaction between a human hand and an object can create rich contact information in reality, which can be considered as continuous at the macro level. Simulating such continuity could be a challenging task. In addition, human hand is deformable on surface and flexible in terms of joint connection. Jain and Liu (2011a) applied soft tissue deformation only at the site of contact, and concluded that soft contact provides robustness and is important to produce more realistic motion. Dexterous hand manipulation tasks can involve fast motion and thin, delicate objects. Lan et al. (2022) combined projective dynamics and the IPC to simulates complicated models on the GPU at an interactive rate or even in real time.

2.2 Control Algorithm and Learning

Control algorithm drives the hand model to complete desired task in physical simulation, and can consist of multiple layers. Witkin and Kass (1988) proposed that by specifying the desired tasks as a series of spacetime constraints, the control can be treated as an optimization problem. ElKoura and Singh (2003) proposed a data-driven approach to reconstruct sympathetic hand posture, and they used this approach to build a procedural virtual guitar player. Kry and Pai (2006) captured motion and contact forces altogether for hand-object interactions, estimated the reference trajectory and joint compliance from captured data, and then resynthesised the detailed hand motion in physical simulation. Ye and Liu (2012) synthesized detailed hand movements of the captured motion of full-

body and object of interest, by randomly tree-searching a set of feasible contact point trajectories and then reconstructing their hand motions. Mordatch et al. (2012a,b) used contact invariant optimization to synthesize complex human full-body behavior and hand manipulation. Kim et al. (2021) added modulated assistive forces to enhance the performance of optimization, and the resulting control policy can even work with the assistive forces. Smith et al. (2020) combined motion capture and physical simulation, and can track complex hand gesture with high fidelity using 43 or more cameras. Li et al. (2022b) used graph convolutional network with pyramid image feature attention module and cross hand attention module to reconstruct two interacting hands from a single image. Lee et al. (2019) build a comprehensive musculoskeletal model which encompasses a full-body musculoskeletal structure with 346 muscles, enabling the simulation of a wide spectrum of human movements under various anatomical conditions, such as differences in bone geometry, muscle strength, and flexibility. It is possible to tailor and apply this model to local structures like human hands.

Learning based methods have recently gained much popularity in simulation and control. Peng et al. (2018) adapted reinforcement learning methods to learn robust control policies that can imitate a variety of input sample motions while achieving user-specified goals. Juravsky et al. (2022) further integrated natural language processing to develop a language-directed controller for character animation using an adversarial imitation learning approach. Zhang et al. (2021) presented a system to synthesize animation of hand manipulating various virtual objects in real-time with a new hand-object spatial representation. On each frame, they fed current frame finger pose, hand trajectory, and values of various virtual sensors into their trained neuron network ManipNet to predict a new finger pose for the next frame as an auto-regressive model. Chen et al. (2022) used a model-free reinforcement learning to learn reorienting objects using a robotic hand under physical simulation. Their policy directly takes a point-cloud observation as input and then outputs the incremental action to the current finger pose. They claimed that it is

possible to learn control strategies for general in-hand object re-orientation that are shape-agnostic. However, most learning based method typically requires a large collection of training data. Inspired by Goodfellow et al. (2020), many works are using generative adversarial networks (GAN) to generate more sample motions from limited collection. Li et al. (2022a) presented a generative model that can learn from a single motion sequence. Xu and Karamouzas (2021) presented a GAN-like method that enables interactive control tasks. Meanwhile, autoencoders provide an elegant approach to reduce the dimensionality. Starke et al. (2022) used a periodic autoencoder to learn periodic features from large unstructured data collection.

2.3 Hand Animation

In terms of the overall system of generating hand animation, synthesis of complex hand motions has been studied by several previous works in computer graphics. Chemin and Lee (2018) use a reinforcement learning approach to train 2D physics-based characters to juggle multiple balls. Their controller relies on continually switching between throwing and catching modes. Earlier work by Jain and Liu (2009) used an optimization framework to generate physically plausible trajectories for objects that match character motion, such as juggling. Their approach is efficient enough for interactive editing of the trajectories, yet real-time control in a dynamics setting was not demonstrated. Yeo et al. (2012) combine gaze tracking with hand, head, and upper body control to generate plausible hand-eye coordination during catching tasks. Other work has demonstrated that low-dimensional feedback controllers are often sufficient for performing simple ball hitting tasks Ding et al. (2015). Dribbling basketballs has been studied by Liu and Hodgins (2018), which requires precise and agile control of the fingers. They propose learning the arm and locomotion control separately, where the arm controller is responsible for hand position and manipulation of the ball. Their approach combines trajectory optimization with reinforcement learning to learn a robust control policy.

Grasping and hand animation is a related topic that has been extensively studied in computer graphics, and Wheatland et al. (2015) provide an excellent survey. Ye and Liu (2012) synthesized realistic hand animations that matched body and object motions captured by an optical marker system while satisfying frictional contact constraints. Pollard and Zordan (2005) synthesized grasping and handshake animations using a a finite state machine that controlled the target pose. Andrews and Kry (2013) similarly based their controller on a finite state machine, but learned control policies for a variety of in-hand manipulation tasks. Liu (2009) performed dexterous manipulation of objects given an initial grasping pose and a desired object trajectory.

Simulation of detailed hand motion like magic tricks has rarely been studied. Li et al. (2020, 2021) used incremental potential contact (IPC), to provide intersection- and inversion-free at large time step, and showed an example of perfect riffle shuffling, using hard-coded application of forces, but without the involvement of hands.

2.4 Works in Robotics and Neuroscience

Hand manipulation have also been studied in robotics. Kober et al. (2012b) learn a catching control policy for a robotic apparatus with a net. They further demonstrate that a control policy for hitting a ball can handily be transformed into one for catching a ball. Kober et al. (2012a) perform robotic catching and juggling using a state machine to open and close the gripper. Their approach also benefits from an approach that combines vision-based tracking and inverse kinematics. Related work by Kim et al. (2014) uses a Gaussian mixture model to plan grasping poses for catching. Belousov et al. (2016) demonstrate that a robust catching policy alternates between reactive and predictive strategies based on the amount of observation noise. Lampariello et al. (2011) compute control parameters offline for many different catching scenarios using a constrained optimization framework, and real-time control is then realized by regression to estimate optimal control parameters.

Work in the neuroscience community has analyzed human arm motion for catching tasks to identify district phases Kajikawa et al. (1999). A key insight is that the motion depends on whether or not an object is being caught with caution, and the catching motion may vary according to the velocity of the object. Salehian et al. (2016) leverage these insights to perform "soft" catching control, which improves the success rate of grasping fast moving objects. The state machine used by our controller also imitates the phases of catching observed by Kajikawa et al. (1999). Additionally, they noted straight trajectories were used for movement of the palm in pre-catching phases, and our controller uses a similar strategy to compute trajectories for the palm based on predicted object motion.

2.5 Summary

Although many works exist in the relevant fields, simulation of detailed hand motion like magic tricks has rarely been studied. In our works, in contrast to all the works discussed above, we do not target robots or build on known neuromuscular control models. Instead, fine adjustments necessary for successful motion control are either learned starting from an easily specified nominal plan for the control or optimized using a reward function. In addition, we stress the involvement of the fingers, because we believe the posture of fingers, the orientation of the palm, and the overall coordination of all joints are critical to synthesizing successful realistic hand motions. Exceptionally, our catching and throwing simulation runs in real-time at 60 frames per second.

3

Methodology

To approach the target of animating hand doing complex jobs, we attempted two different paths as explained in the following two sections. First, is a nominal controller assisted by reinforcement learning synthesizing catching and throwing different objects. Second, is an optimization based controller synthesizing manipulation of cards by fingertips. Note that they are simulated in different setups as for achieving different goals. This is because we found that during our experiments the reinforcement learning-assisted controller tends to handle dynamic and repetitive tasks more effectively, while the optimization-based controller can often achieve higher precision in delicate simulation scenarios.

3.1 Catching and Throwing with RL

We design a nominal controller assisted by reinforcement learning for animating an articulated physics-based human arm model, including the hands and fingers, to catch and

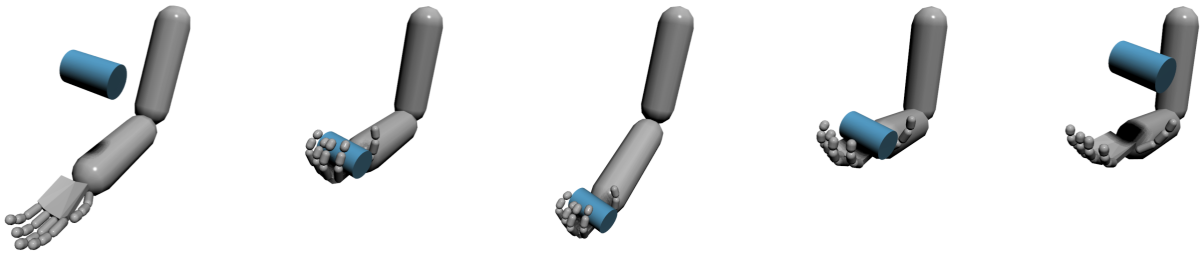


Figure 3.1: Example of catching and throwing with reinforcement learning.

throw objects as shown in Figure 3.1. The controller is based on a finite state machine that defines the target poses for proportional-derivative control of the hand, as well as the orientation and position of the center of the palm using the solution of an inverse kinematics solver. We then use reinforcement learning to train agents to improve the robustness of the nominal controller for achieving many different goals. Imitation learning based on trajectories output by a numerical optimization is used to accelerate the training process. The success of our controllers is demonstrated by a variety of throwing and catching tasks, including flipping objects, hitting targets, and throwing objects to a desired height, and for several different objects, such as cans, spheres, and rods.

3.1.1 Hand and Arm Modeling

For simulating catching and throwing, our hand model includes a fully articulated arm and hand with 32 degrees of freedom (DOF) in total, as shown in Figure 3.2 where joints are represented by line segments with different colors. The upper arm is connected to a fixed shoulder (not shown in the figure) with a ball joint. The angular limits and anchor points of the joints are similar to that of human joints. The shape, size, and mass of the model are based on the 50^{th} percentile of American male NASA (1995). Following the work of Pollard and Zordan (2005), we use a simple mesh to approximate collisions with the human hand for a better grasp (i.e., the cup-like shape of the palm) which is a common practice for real-time simulation with contact as suggested in Bender et al. (2014). The motion of the whole model is driven relative to a neutral pose, which provides good parametric control of joints over a range of motion suitable for catching.

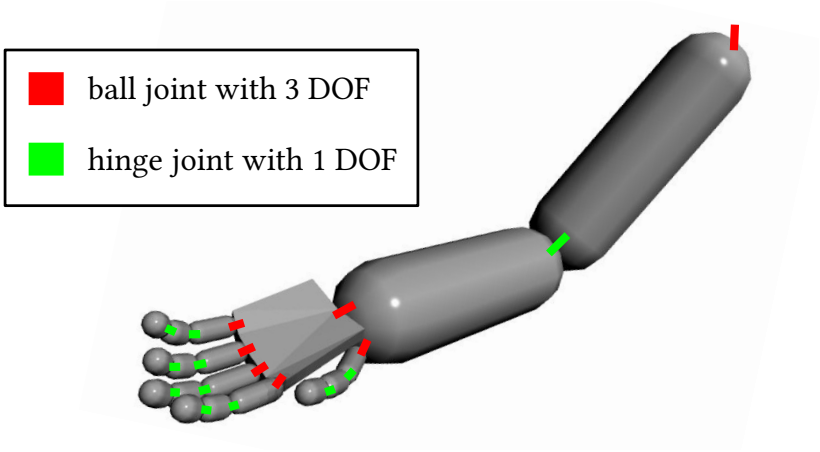


Figure 3.2: Arm and hand with fingers in its neutral pose.

3.1.2 Nominal Controller Structure

The scheme of nominal controller is defined as a finite state machine as shown in Figure 3.3. This state machine determines the trajectory of the center of the palm (position and orientation) at each frame, as well as the timing for opening and closing the hand. The palm trajectory is generated by a Hermite curve, which is then tracked using IK. Whereas the hand posture is tracked using PD (proportional-derivative) control and a set of predefined poses. PD control is a type of feedback control system commonly used in engineering and robotics. It applies torques or forces based on its two components. Its proportional control component reacts to the current error, which is the difference between the desired target state and the current state. The derivative control component reacts to the rate of change of the error, for example, the velocity of the object under control. A detailed explanation of PD control is given in Section 3.1.5.

The FSM consists of four states:

- `CATCH`: The hand opens and moves to the interception point to intercept the projectile at time *hit_time*, which is dynamically calculated by the nominal controller for each catch. The hand closing is triggered by the first collision between the hand and the projectile since the beginning of each `CATCH` state, either in this state or in the following `MIDDLE` state.

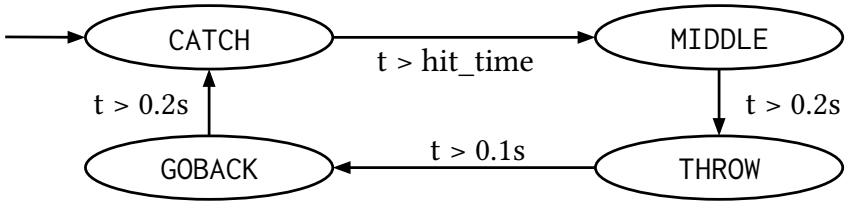


Figure 3.3: The finite state machine used by our controllers.

- `MIDDLE`: The hand decelerates to zero velocity while moving the projectile to the neutral position, which is suitable for throwing the projectile.
- `THROW`: The hand throws the projectile. See Section 3.1.8.
- `GOBACK`: The hand goes back to a position which is suitable for the next catch. See Section 3.1.7.

The timing of the transition between states is defined as in Figure 3.3, where t is the elapsed time since the beginning of each state. These values are based on our intuition of how humans handle projectiles, and they were adjusted to produce natural motion for simple cases, such as catching and throwing a ball.

Note that the Hermite curves specify the trajectory of the center of the palm, and inverse kinematics (Section 3.1.4) is used to to solve for the corresponding joint angles of the arm. All parameters defining the curves for each state are either specified by the nominal controller or computed by reinforcement learning. Rotational motion is generated using spherical linear interpolation (SLERP) between the beginning and ending rotations of each state. Most target rotations are specified by the nominal controller, except for the final rotation of the palm during the `THROW` state, which is determined by an RL agent. Similarly, the timing of opening the hand during `THROW` is a parameter that is automatically adjusted by the RL algorithm, which is explained in more detail later in Section 3.1.8. Conversely, during the `CATCH` and `MIDDLE` states, the hand closes when contact between the hand and the projectile is detected.

3.1.3 Palm Trajectory

In each state, we calculate a trajectory for the palm of the hand, which is defined by a Hermite curve that interpolates the position, orientation, and velocity of the palm from an initial configuration to the target configuration at the end of the state. PD control is used to actuate the joints of the arm and wrist in order to drive the hand to that target. In order to ensure a smooth and natural motion for the arm, intermediate PD joint angle targets are computed by interpolating along the Hermite curve for the duration of the state.

At each time-step, an intermediate target configuration for the palm is computed in the time interval $[t_0, t_1]$, where t_0 is the time when the controller transitioned to the current state, and t_1 is the time when the controller will transition to the next state.

The intermediate position of the palm $p(t)$ at some time $t \in [t_0, t_1]$ is thus computed as

$$\begin{aligned} p(t) = (2t^3 - 3t^2 + 1)p_0 + \Delta t(t^3 - 2t^2 + t)v_0 \\ + (-2t^3 + 3t^2)p_1 + \Delta t(t^3 - t^2)v_1 \,, \end{aligned} \tag{3.1}$$

where $\Delta t = (t_1 - t_0)$, p_0 is the initial position of the palm, v_0 is the initial velocity, p_1 is the ending position, v_1 is the ending velocity, and $p(t)$ gives the desired position at time $t \in [t_0, t_1]$. However, for the orientation of the palm, SLERP is used to compute an intermediate rotation.

The Hermite curve ensures that both the position and velocity of the hand are continuous during the time interval. For the `CATCH`, `MIDDLE`, and `GOBACK` states, the planning algorithm computes the parameters of the Hermite curve according to the position, orientation, linear and angular velocity of the projectiles. Further details on catch planning are discussed in Section 3.1.7. For the `THROW` state, the parameters are controlled by the reinforcement learning agent and further details are discussed in Section 3.1.8.

3.1.4 Inverse Kinematics

The spline and SLERP interpolations compute the target position and orientation of the palm at any time t. Our controller then uses a damped least squares IK algorithm Buss and Kim (2005) to solve for the joint angles of the shoulder, elbow, and wrist joints. This gives an update for the joint angles at each time step, such that

$$\Delta\theta = J^T(JJ^T + \lambda^2 I)^{-1}\Delta e\,, \tag{3.2}$$

where J is the end effector Jacobian matrix, I is a identity matrix, λ is a non-zero damping constant, and Δe is a vector encoding the transform from the current palm configuration to the target configuration. Note that the position and orientation of the end effector (i.e., the center of the palm) are weighted differently in the Jacobian matrix J for better catching. Hence, $\Delta\theta$ gives the changes for the joint angles to track the desired trajectory of the palm.

3.1.5 PD Control

Proportional-derivative (PD) control is used to perform low-level control of the arm and hand by computing joint torques that actuate the joints toward the desired pose. At each time step, a torque is computed for each articulated degree of freedom i, such that

$$\tau_i = k_p \underbrace{\left(\tilde{\theta}_i - \theta_i\right)}_{\Delta\theta_i} + k_d\dot{\theta}_i, \tag{3.3}$$

where k_p is the joint stiffness, k_d is the damping of the joint, $\tilde{\theta}_i$ and θ_i are the target angle and current angle of the degree of freedom, respectively, and $\dot{\theta}$ is the relative angular velocity of the joint. The control torque τ is then applied equally and oppositely to bodies coupled by the joint. Furthermore, the torques are clamped so that the value does not exceed typical human strength.

For every joint, the k_p and k_c is calculated by

$$k_p = w_p \, {m_{\text{joint}}}^2 + k_{\text{base}} \,, \qquad k_d = w_d \, m_{\text{joint}} \,, \tag{3.4}$$

where w_p and w_d are constant scaling factors for the stiffness and damping, respectively, m_{joint} is the total mass driven by the joint, and k_{base} is a constant base stiffness. The values $w_p = 500$, $w_d = 0.2$, and $k_{\text{base}} = 0.1$ are used for all experiments, but they can be easily adjusted to simulate different muscle strengths. We use PD control on IK results smoothly interpolated by Hermite curves. And from the intuition that the hand grasp is tighter as it closes harder, we set the PD target of a closing hand to fit a smaller shape than the projectile. Additionally, we expect the learning process should compensate the inaccuracy of PD.

3.1.6 Hand Pose

We found that performance is improved by applying the IK control only for positioning the joints from the shoulder to the wrist. This gave sufficient control from positioning the wrist, whereas the posture of the fingers and thumb were more effectively controlled by using a set of predefined poses. That is, the target hand pose is one of three predefined poses: neutral, open, and closed. The neutral pose is used during the `GOBACK` phase. The closed pose is used for the `CATCH` and `MIDDLE` phases of the controller, whereas the hand model transitions to the open pose for the `THROW` phase. To smoothly transition between the different hand postures, we use SLERP to generate intermediate joint angle targets for the PD controllers of the fingers to achieve natural opening or closing motions.

3.1.7 Trajectory Planning

Given the initial position and velocity of the projectile, our controller predicts the trajectory of the projectile. The intersection of the trajectory and the interception plane (the plane where we want the hand to catch the projectile) is the target intersection point. For

the shapes whose orientation also matters, we also take the orientation into account. After we find the interception point, the end position of the Hermite curve for the `CATCH` state is simply the interception point. The end velocity of the curve is set by the nominal controller to 70% of the velocity of the projection at the interception point, which is intuitively similar to human catching behavior and makes the projectile less likely to slip away. The start position and velocity of the curve are the position and velocity of the hand when the controller just enters the `CATCH` phase. As for the orientation, our controller aligns the normal of the dorsal side of the hand with the velocity of the projectile at the interception point. Additionally, for projectiles similar to can or club, our controller aligns the z-axis of the hand (as in the right-hand rule) with the longest axis of the projectile at the interception point of the catch.

For the `MIDDLE` state, the end position is the neutral position while the end velocity is zero. For the `GOBACK` state, the end position is the projection of the hand position on the rest plane ($y = 0.6$ m) when the controller just enters the `GOBACK` state. The end velocity is also zero. The start positions and velocities of curves are the positions and velocities of the hand when the controller enters these states.

3.1.8 Reinforcement Learning

We use a reinforcement learning agent to control the throwing process, for example, the parameters of the Hermite curve of the `THROW` state. The state vector is (p_h, r_h, p_p, r_p), where p_h and r_h is the hand position and rotation vector in the world frame, p_p is the object position in the hand's frame, and r_p is object rotation vector in the world frame. The action vector is $(d_{\text{offset}}, r_{\text{target}}, t_{\text{open}})$, where d_{offset} controls the throwing direction, r_{target} controls the hand rotation, and t_{open} is the time when opening the hand. The output (p_h, r_h, p_p, r_p) is sent to the nominal controller to plan the trajectory and set the PD targets for the hand and arm. The reward function varies with different task (see Section 4.1).

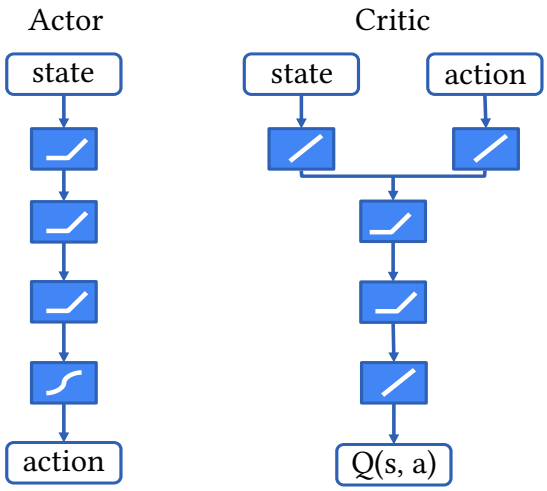
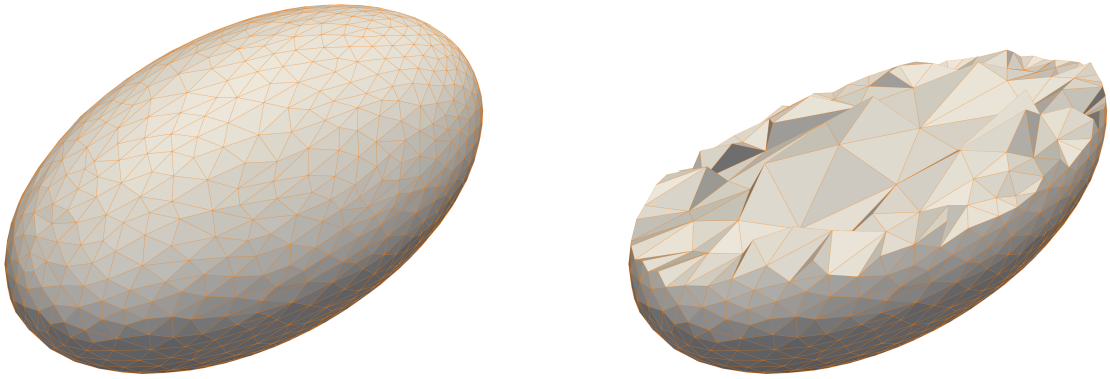


Figure 3.4: The actor-critic network.

We use deep deterministic policy gradient (DDPG) by Lillicrap et al. (2019) to train our agent. DDPG is chosen because it is well-suited for problems with continuous action spaces as we have here. Furthermore, its model-free, off-policy approach enhances the sample efficiency and allows learning from experiences. The structure of the actor and critic networks is shown in Figure 3.4. Each fully connected layer has 64 nodes. We use ReLU activation as the activation function except for the output of the action network, where we use $\tanh$ as the activation function.

3.2 Card Manipulation with Optimization

We develop an optimization-based controller to simulate card manipulation tasks done by soft fingertips. The fingertip is modeled as a tetrahedral mesh and simulated via the Finite Element Method (FEM), while the cards are represented as two-dimensional thin shells. These thin shells and tetrahedral meshes are coupled using the Codim-IPC (codimensional incremental potential contact by Li et al. (2021)) to enhance interaction fidelity. A bone-based controller is employed to drive the central parts of the hand model, effectively simulating the movement of human bones and the consequent driving of surrounding soft tissues. The controller's parameters are optimized using MOTPE (Multi-Objective Tree-structured Parzen Estimator by Ozaki et al. (2020)), with objective func-



(a) Surface of the fingertip. (b) Clip of the fingertip.

Figure 3.5: Tetrahedral fingertip example.

tions that captures the dynamic state of the cards at each step of manipulation, ensuring realistic and precise animation outputs.

3.2.1 Fingertip and Card Modeling

For simulating card manipulation, we use tetrahedral mesh to model the fingertips which directly interact with the cards. The fingertips are tetrahedral ellipsoid as shown in Figure 3.5, where each fingertip consists of 6851 cells and 1760 points. The cards are modeled as 2-dimensional triangulated grids with rounded corners as shown in Figure 3.6, where each card consists of 141 triangles and 93 vertices in total. A regular pile of poker cards has 54 such cards perfectly aligned and stacked together.

3.2.2 Codimensional Incremental Potential Contact

To address the coupling of 3-dimensional hands and 2-dimensional cards, we adapt the Codim-IPC by Li et al. (2021). Interaction of hands and cards involves a large number of contacts between objects of different dimensions, especially 2-dimensional thin shells and 3-dimensional hands. Traditional methods often struggle with these interactions by inducing numerical instability or inaccuracies, which can lead to interpenetration between object or unnatural contact forces on surfaces. Codim-IPC addresses these issues by incre-

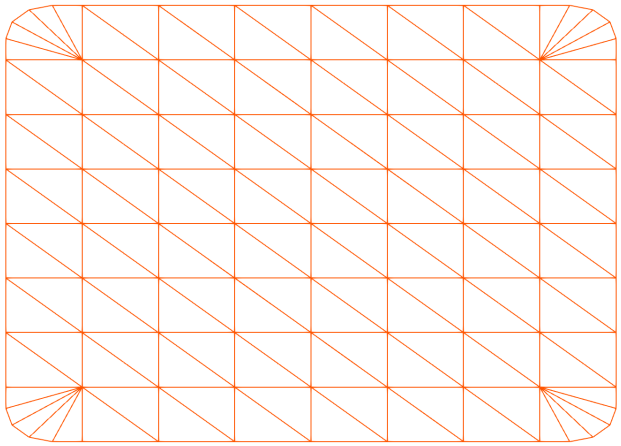


Figure 3.6: Regular card triangulation example.

mentally updating the simulation's potential energy of all types with their barrier contact model and additive continuous collision detection method.

3.2.3 Bone-based Control

Different from controlling the movement of rigid bodies in the Section 3.1 where we apply torques directly on rigid parts to make them rotate around their respective joints as a whole, controlling the soft bodies requires some portions of the bodies remain soft to deform when having contacts with the other bodies. This can be solved by applying a moving constraint on a set of chosen points inside the soft bodies. We use Dirichlet boundary condition to drive the inner parts of the hands as rigid bones doing rigid movements as in the Section 3.1 while maintaining the superficial parts soft as muscles and skins to interact with the soft thin-shell cards.

3.2.4 Optimizing the Control

In a most straight forward way, we can optimize a complete magic trick as a whole where the decision variables are the trajectories of bones and the multi-objective function specifies the differences from target card configuration on each checkpoint where the card

configuration matters. However, this is almost impossible considering the size of problem. As mentioned in Chapter 1, magic trick can be decomposed into multiple sleights. Hence, for each individual sleight, we tailor the decision variables and objective function to fit the specific goal card configuration. That is, each sleight starts with an initial configuration of hand and cards, the decision variables are the trajectories of bones and the objective function describes the difference from the end card configuration to the target card configuration after the expected time to finish the sleight elapsed in the simulation.

Considering the expensive cost of Codim-IPC simulation with complex geometries, we use the MOTPE to optimize the bone-based control of hand, because the MOTPE is designed to search the high-dimensional space efficiently using a tree-structured Parzen estimator to provide promising solutions, and generally requires less trails comparing the other optimization methods. In addition, MOTPE can also provide a list of best trials for exploring multiple ways to perform the task.

In addition, similar to the catching and throwing controller problem, to leverage the high cost of dealing with the high dimensionality during optimization, we set constrains for the decision variables by making a reasonable guess of its possible range. That is, when snapping cards apart, the finger should not release the card by moving in the direction perpendicular to the card surface, and when lifting cards, the upward component of the average velocity of the fingertips must be positive and no faster beyond human ability. Hence, by providing reasonable domains for the decision variable, the optimization via MOTPE can be fast and reliable.

3.3 Discussion on Methodology

In this chapter, we present the methodologies of our two systems for animating a human hand performing various tasks in physical simulations. Our first system employs a finite state machine and PD control to manage the hand's movements, with reinforcement learning further enhancing the controller's robustness and adaptability for various tasks.

Our second system uses a bone-based controller, optimized via the MOTPE, to obtain detailed control parameters that ensure effective card manipulation. Although the two systems are constructed upon different simulation settings and object models, they share the idea of allowing the computer to automatically generate control parameters, which are usually hard or time-consuming for humans to work on, in synthesizing hand-object interactions.

4

Result

Here we present the various results for the two systems introduced in the Chapter 3. In the following sections, we report the detailed parameters for simulation, reward functions or objective functions for each scenario, error analysis, learning curves, optimization statistics, and illustrations of the motions generated by our controllers.

4.1 Catching and Throwing with RL

For the catching and throwing, we present three scenarios in which our controller learns a desired catching and throwing motion: throwing to a desired height, throwing to hit a target, and flipping an object. Below we describe the simulation environment, and the reward necessary for each scenario. Our controller can do the catch-and-throw in consecutive loops just like in the acrobatics. And the reinforcement learning greatly improves the performance of our controller. Starting from the same initial configuration on the same

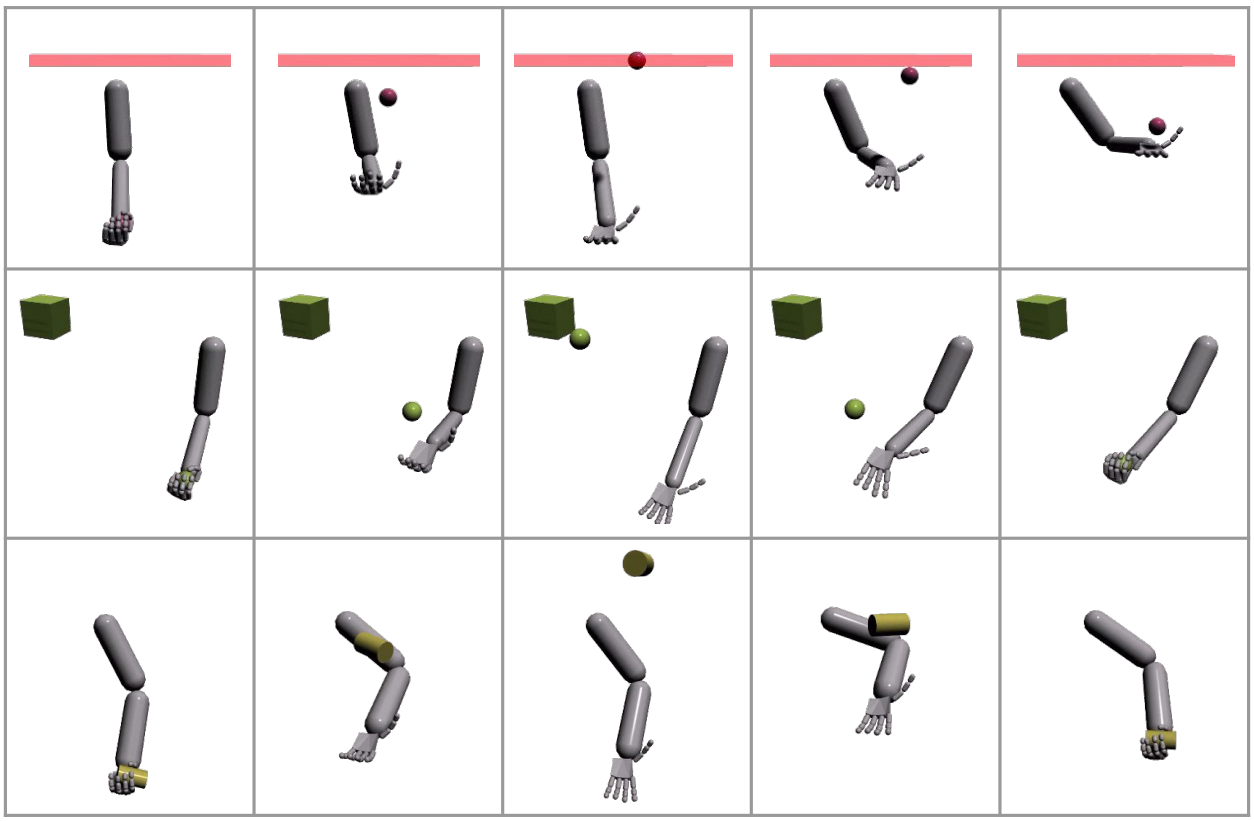


Figure 4.1: Example motions produced by our controller.

task, the nominal controller with hand-tuned parameters usually fails at the 2-5$^{\text{th}}$ loop of catch-and-throw, but the RL controller with learned parameters appear to be able to loop infinitely. The example motions produced by our controller are shown in Figure 4.1 line by line and the details of these motions will be explained in their respective sections.

4.1.1 Rigid Body Simulation

Physics simulations are performed using ode4j, which is a java port of the Open Dynamic Engine. All experiments use a time step of $h = 0.2$ ms , which is the same as Pollard and Zordan (2005). We found that is was necessary to carefully tune simulation parameters in order for successful grasping of ballistic objects. Specifically, the error reduction parameter (ERP) and constraint force mixing (CFM) term, which are used by the engine for Baumgarte constraint stabilization, were tuned to give the behavior of compliant contacts. We note that other researchers have also noted the importance of compliant contacts for

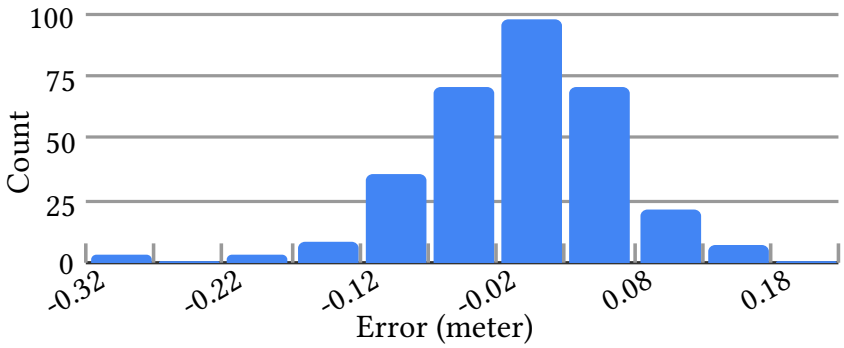


Figure 4.2: Error distribution of catching and throwing a sphere to the desired height.

successful grasping of objects Jain and Liu (2011b). A stiffness coefficient of $k_s = 5 \times 10^4$ was used for all contact constraints. A critical damping coefficient was also computed, such that $k_c = 2\sqrt{k_s(m_1 + m_2)}$ where m_1 and m_2 are the masses of the two colliding bodies. The ERP and CFM parameters are then computed as

$$ERP = \frac{hk_s}{hk_s + k_c}, \qquad CFM = \frac{1}{hk_s + k_c}. \tag{4.1}$$

The simulator resolves collision by creating a temporary contact joint which applies forces to the two colliding bodies. The contact joint is used to detect collisions between the hand and projectile as mentioned in Section 3.1.2.

Additionally, a small amount of world damping is applied to the linear and angular velocities of all bodies in order to model air resistance. A linear velocity damping of 0.01 and angular damping of 0.03 were used for all of our simulations. We also observed that damping helped to reduce oscillations that can occur due to stiff PD controllers.

4.1.2 Throwing to Desired Heights

Our first example shows throwing a sphere to the desired height, and then catching the sphere and repeating this process with different heights. The sphere weights 0.1 kg and its radius is 0.03 m. The desired height changes over time. In our implementation, we embed the goal to the state vector. In this example, it is the desired height y_{target}. The

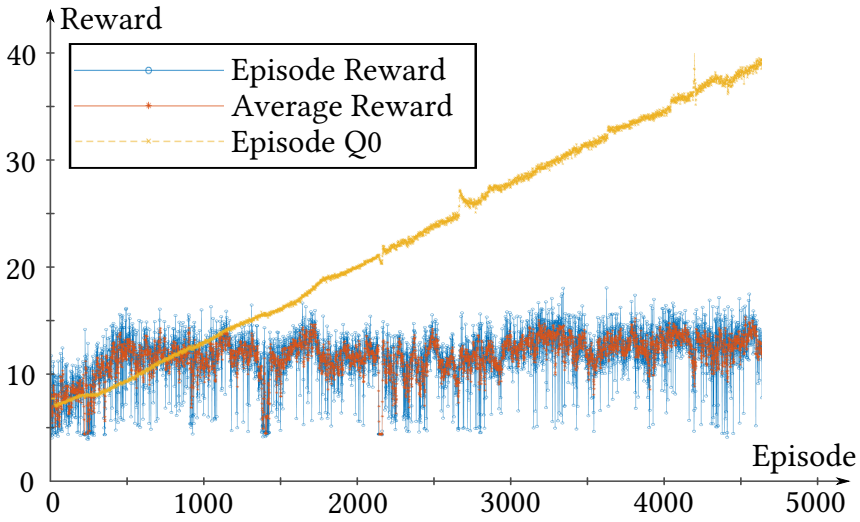


Figure 4.3: Learning curve of throwing to a desired height.

reward function we use is

$$r = \begin{cases} e^{10|y - y_{\text{target}}|} & \text{if the hand catches the sphere} \\ -0.1\,|p_p - p_d| & \text{if the hand misses the sphere} \end{cases} \tag{4.2}$$

where y is the maximum height the sphere reaches, y_{target} is the desired height, p_p is the position of the sphere when it hits the ground, and p_d is the target position to prevent the hand from throwing the object far away from the hand. We only use the error term $y - y_{\text{target}}$ because we want to learn how to deal with different heights. Note that the ground needs to be explicitly defined here to determine if the hand has missed the sphere. That is, when the sphere goes beyond the reach, the simulation should terminate and report a miss.

Before learning, the nominal controller has to be hand-tuned for every different height. With the help of reinforcement learning, the controller can catch and throw the sphere to different heights with learned parameters in real-time. Figure 4.2 shows the error distribution in 320 consecutive catch-and-throw loops with learned parameters, where the desired height is randomly set in each loop. The slightly left skewed error distribution with mean around 0.01 m shows that the learned controller can generally achieve the task

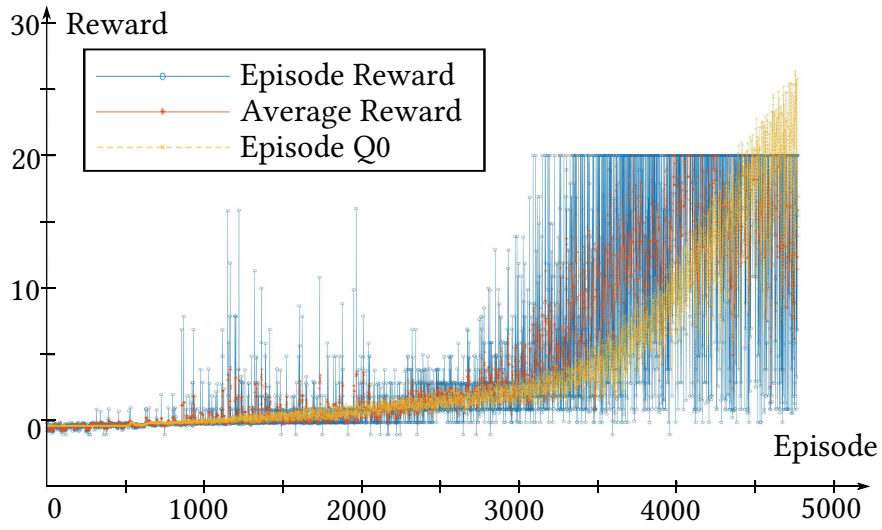


Figure 4.4: Learning curve of catching and throwing the sphere to hit a target box.

within 0.1 m of error. Figure 4.3 shows the learning curve, where the episode and average reward fluctuates around 10 and the Episode Q0 increases linearly. Here, the Episode Q0 is the estimate of the discounted long-term reward at the start of each episode, given the initial observation of the environment. Top row of Figure 4.1 shows the visual result where the target height is indicated by a transparent red plane.

4.1.3 Throwing to Hit Target

The second example shows throwing a sphere to hit a box, catching it when it bounces back, and repeating this process. The box has a size of 0.1 m $\times$ 0.1 m $\times$ 0.1 m. The position of the box is encoded into the state vector in order to hit the box in different positions. The reward function we define is

$$r = \begin{cases} 1 & \text{if "hit then catch" is successful} \\ -0.1\,d & \text{otherwise} \end{cases} \tag{4.3}$$

where d is the minimum distance between the sphere and the cube. Note that a successful hit and catch is rewarded by a constant here because the point of hit does not matter in this setup.

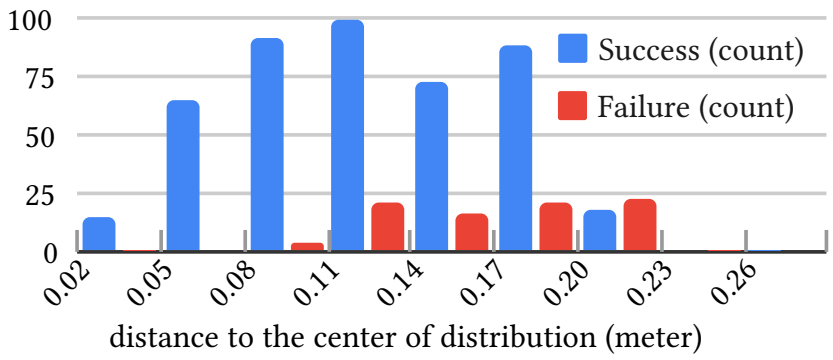


Figure 4.5: Distance to the center for the position of the box where the agent succeed in or fail the hitting box task.

Before learning, the nominal controller has to be hand-tuned for every different box position. After learning, the agent can throw the sphere to hit boxes with different position in real-time. Figure 4.4 shows the learning curve, where the learning speeds up around the 3000^{th} episode but the rewards greatly fluctuate. Middle row of Figure 4.1 shows the visual result. Event though the agent can hit the box at many locations, there are still some positions where the agent fails to hit the box or cannot catch the sphere when it rebounds, especially near the boundary. We test the agent by setting the target box to different positions across the space. Each test is considered successful if the agent can consecutively hit a fixed box 6 times. Figure 4.5 shows the distribution of successes and failures in 538 tests. Generally, as the box goes further from the center, the rate of success decreases.

4.1.4 Can Flipping

Our third example shows flipping a can over and over. The can weights 0.1 kg. The radius of the can is 0.03 m and the height is 0.1 m. To encourage flipping, we define the reward function as

$$r = \begin{cases} e^{10|p_p - p_d|} + r_{\text{flip}} & \text{if "flip then catch" is successful} \\ -0.1\,|p_p - p_d| & \text{otherwise} \end{cases} \tag{4.4}$$

where r_{flip} is 1 if there is a flip, and 0 if there is not. We use the idea of imitation learning in order to speed up the training process. In detail, we use covariance matrix adaptation

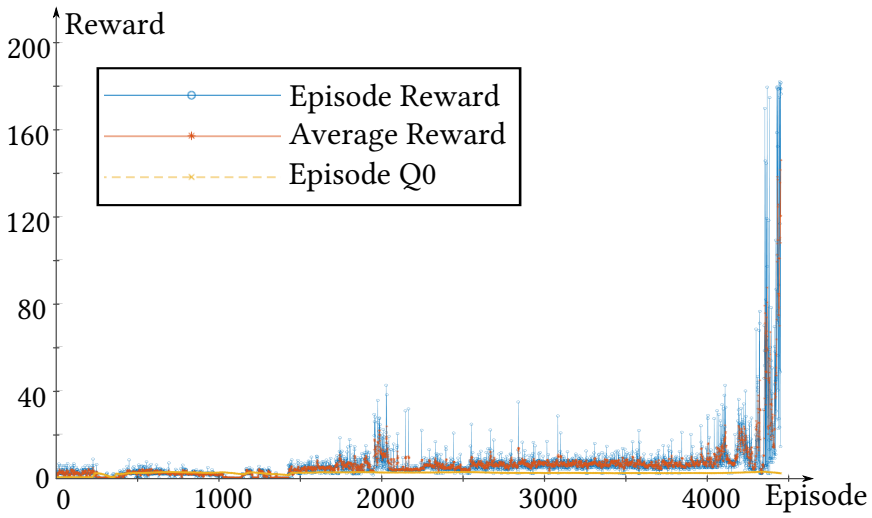


Figure 4.6: Learning curve of flipping the can.

evolution strategy Hansen (2006) to learn how to make a single flip with different initial conditions. Then we use these examples to train an actor-network, which we use as the initial actor-network for the agent.

Before learning, the nominal controller usually fails around the 2^{nd} to 5^{th} flip. With the help of reinforcement learning, the controller appears to be able to do the flipping infinitely. The result animation can be seen in the bottom row of Figure 4.1, while the learning curve can be seen in Figure 4.6.

4.2 Card Manipulation with Optimization

For the card manipulation, we present two scenarios in which we optimize the fingertips to do a desired card sleight: Card Snapping and Double Lift. Below, we discuss the necessary components for a successful simulation of card manipulation: the simulation environment, the initial conditions, the discretization of cards, the decision variables, and the objective functions. Additionally, we interpret and analyze the optimization statistics and the simulation results.

4.2.1 Soft Body Simulation

Setting up the soft body simulation properly according to their respective physical properties in real world is the cornerstone of card manipulation. The cards are set to thin shells made of 100% polyester (PES), which is the most common material of playing cards. And the hand is set to have a density of 1500 kg/m^3, a Young's modulus of 5000 kPa, and a frictional coefficient of 0.4, similar to a ordinary human hand.

4.2.2 Initial Conditions

The stability and runtime of simulation in a single trial greatly depends on the initial conditions. Especially, for a resting card stack of 54 cards, ill-conditioned initial configuration of cards can lead to undesired shifting of cards caused by ghost forces, longer time for the card stack stabilizing itself to static, and even interpenetration between cards. To address this problem, we first set the distance between every two adjacent cards to be the exact thickness of a card plus a small offset ϵ which is 5% of the card thickness, and then warm up the simulation for a few seconds to make sure the card stack come to a total rest. This rest state is saved as a checkpoint and reused at the beginning of every trial. Note that we use the same perfectly aligned card stack reconstructed from the checkpoint in every Double Lift trial. Comparing to a trial without warm-up, a trial starts with the saved warm-up checkpoint is 2 minutes faster.

4.2.3 Discretization

We also explore how the discretization of cards affect the performance of simulation. When two cards of the same regular triangulation as shown in Figure 3.6 are perfectly aligned and stacked together, each vertex on one card may form primitive contact pairs with the aligned vertex on the other card and all the surrounding faces and edges of that aligned vertex, causing numerical perturbation, redundant contact information, and possibly slow-down of simulation. In contrast, when the two aligned cards have different

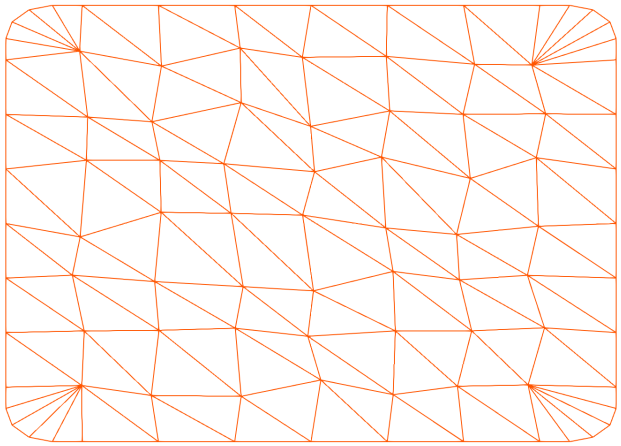
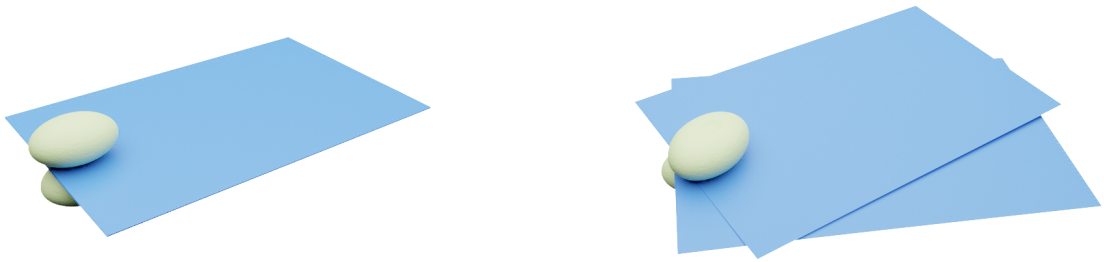


Figure 4.7: Randomized card discretization example.

triangulation, a vertex on one card may only form one contact pair with a face on the other card, or may form contact pairs with an edge on the other card and its two surrounding faces. To alleviate numerical perturbation, enrich the contact information, and possibly speeding up the simulation, we try randomizing the mid-surface vertices of the card on the resting mid-surface plane during the triangulation, except for the boundary vertices, by a small radius as shown in Figure 4.7. However, this method does not provide noticeable speed up in trials. At the mean time, this method may lead to different bending behavior of the cards, especially when bending two aligned cards together. This bending behavior is due to the fact that these edges are not parallel to the borderline edges of card any more. Therefore, randomization on the card triangulation is not adapted in our example scenarios. In the meantime, a uniform, regular triangulation of cards can make the bending and stacking behavior of cards more consistent, which is a desired feature of a set of playing cards in reality.

4.2.4 Card Snapping

The first example of card manipulation shows Card Snapping, which means snapping two cards apart by a desired angle of certain degrees by two or three fingers. Initially,



(a) Initial Configuration (b) End Configuration

Figure 4.8: Card Snapping example.

two cards are perfectly aligned and loosely held by two fingers as shown in Figure 4.8a. And the target end card configuration C_{target} is that the two cards are apart by a desired angle α_{target}. Generally, this requires the two fingers to pinch the cards and then slide them apart. To specify these two action, we set the movement trajectory T of bones inside the two fingers over the time limit t as a combination of pinching in time duration $[t_0, t_1]$ with average velocity $\overline{v_p}$ and sliding in time duration $[t_2, t_3]$ with average velocity $\overline{v_s}$. For optimizing this problem, the decision variables are the movement trajectory T, and the objective function to be minimized is

$$f(C_{\text{end}}) = |\alpha_{\text{target}} - \alpha_{\text{end}}| \tag{4.5}$$

where the C_{end} is the end configuration of cards, α_{end} is the angle between cards. Here, the objective function specifies the angle of snapping. The resulting end configuration is shown in Figure 4.8b, where two cards are snapped apart by an angle.

Figure 4.9 shows the optimization history of 89 trials, where each blue dot represents a single trial. The best objective value quickly get minimized close to zero within 20 trials and stays almost 0 after 46 trials. Hence, the optimization is successful and converges very

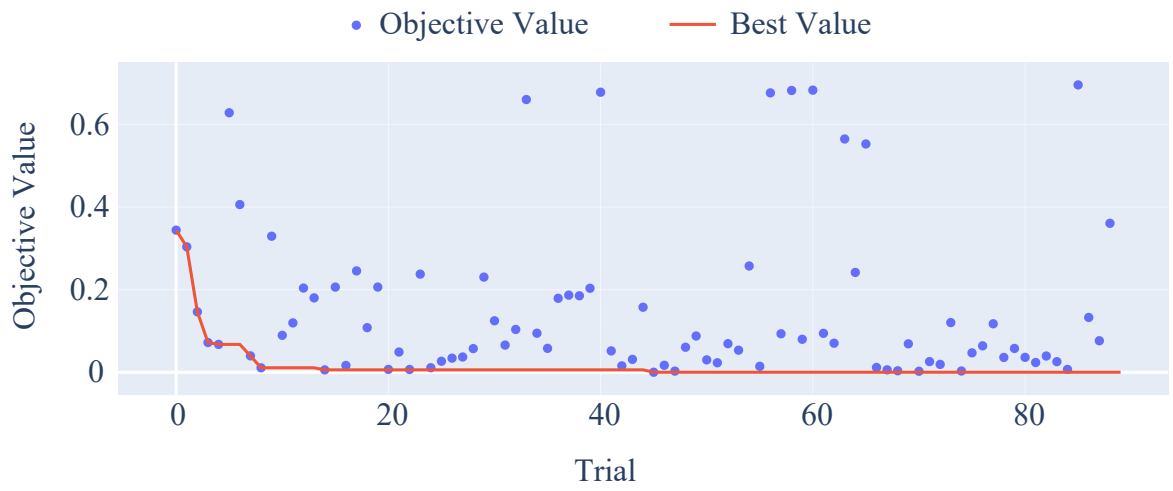


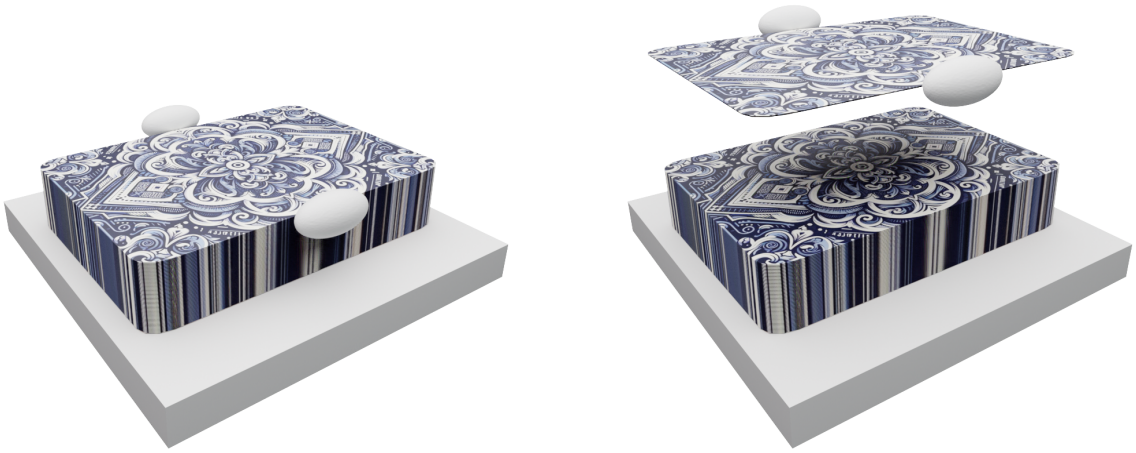
Figure 4.9: Optimization history of optimizing Card Snapping in 89 trials.

fast in terms of the number of trials. Note that some trials still get large objective values due to the sampling strategy of the MOTPE and the discrete nature of cards. Besides, the average runtime for each trial is 41.0 seconds, while the minimum runtime is 30.1 seconds and the maximum runtime is 72.4 seconds as summarized in Table 4.1. The fluctuation of runtime between trials is expected, because the simulation step takes longer when the fingertips pinch the cards harder and more contacts are created.

Note that for this example we use a simplified card which has sharp corners. The sharp corner consist of two triangles instead of five as in Figure 3.6 or Figure 4.7. Surprisingly, the optimization result of snapping using sharp corner cards cannot be applied directly onto the snapping using rounded corner cards. We suspect that this is due to the change of frictional forces. This may worth further investigation.

4.2.5 Double Lift

The second example of card manipulation shows the sleight Double Lift, which means lifting two cards as one. Initially, 54 cards are perfectly aligned and stacked on a static horizontal table and two fingers starts around the top of the card pile one on each side (not touching the cards) as shown in Figure 4.10a. And the target end card configuration C_{target} is that the top two cards are apart from the rest of cards by a certain height h_{target}.



(a) Initial configuration. (b) End configuration.

Figure 4.10: Double Lift example.

Table 4.1: Optimization runtime summary.

Study Name	Trials	Best trial no.	Time avg.	Time min.	Time max.
Card Snapping	89	46	41.0s	30.1s	72.4s
Double Lift	297	152	79.2s	22.9s	808.1s

Generally, this requires the two fingers to pinch two cards on the edges and then lift them as one. To specify these two action, we set the movement trajectory T of bones inside the two fingers over the time limit t as a combination of pinching in time duration $[t_0, t_1]$ with average velocity $\overline{v_p}$ and lifting in time duration $[t_2, t_3]$ with average velocity $\overline{v_s}$. For optimizing this problem, the decision variables are the initial height h_{start} of the two fingers and the movement trajectory T, and the objective functions to be minimized are

$$f(C_{\text{end}}) = (|h_1 - h_{\text{target}}|, |h_2 - h_{\text{target}}|, \|d_3\|) \tag{4.6}$$

where h_1, h_2 are the heights of the top two cards in the end card configuration C_{end}, and d_3 is the displacement of the third card. The resulting end configuration is shown in Figure 4.10b, where two cards are magically lifted as one.

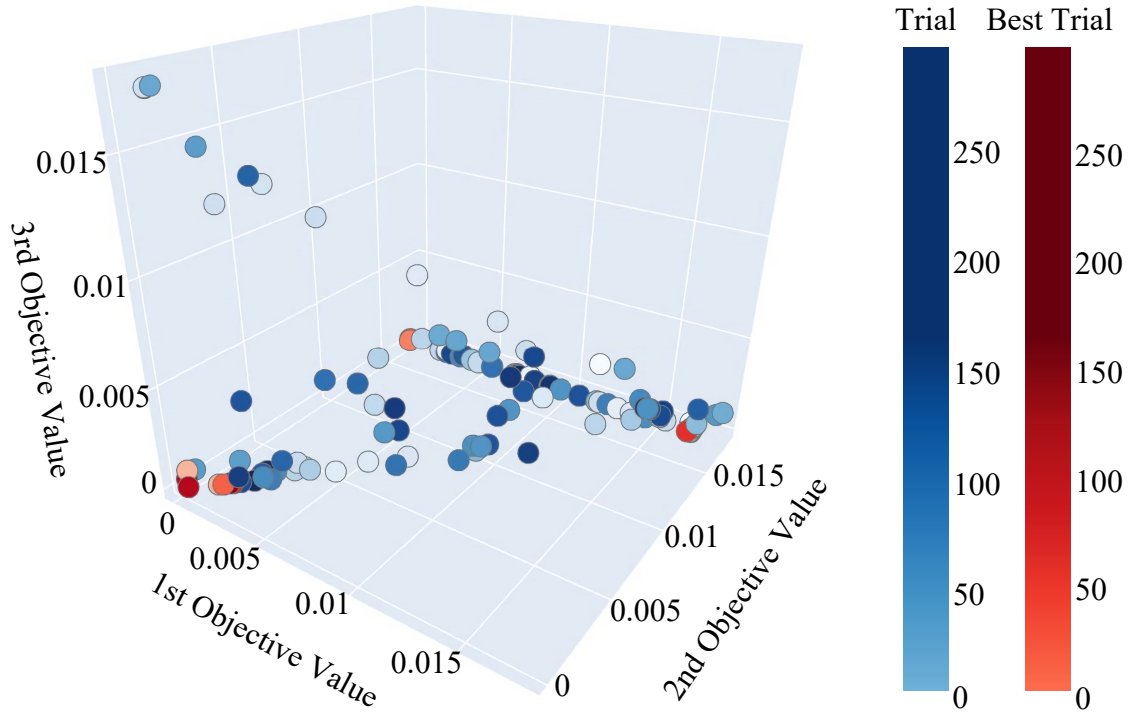


Figure 4.11: Pareto front of optimizing Double Lift in 297 trials.

The Pareto front of optimizing Double Lift is shown in Figure 4.11, where each sphere represents a single trial with its coordinates correspond to its objective values. The red and orange dots represent the best trials that are on the Pareto frontier. The first best trial is trial number 72 and is surrounded by multiple best trials (some are obstructed due to the limitation of showing a 3D graph in 2D). Hence, although the objective functions are simple and non-continuous (because the cards are discrete objects), the optimization is successful and converges fast in terms of the number of trials. Besides, the average runtime for each trial is 79.2 seconds, while the minimum runtime is 22.9 seconds and the maximum runtime is 808.1 seconds as summarized in Table 4.1. The fluctuating runtime is due to the nature of Codim-IPC in simulation thin shell deformables. To find the minimum time step of inversion-free and penetration-free time step, Codim-IPC may search in tiny intervals with tiny stepping which takes a significant amount of time.

4.3 Discussion on Result

In this chapter, we present the results for the two animation systems described in the Chapter 3, focusing on various scenarios involving catching and throwing objects with reinforcement learning and in-hand card manipulation with optimization. For the RL-based system, we detail three tasks: throwing objects to a desired height, hitting a changing target, and flipping a can. We highlight the simulation parameters, reward functions, and performance improvements achieved through reinforcement learning, noting that the learned controllers can consistently perform the tasks with high accuracy and robustness in a loop. In the optimization-based card manipulation, we explore Card Snapping and the Double Lift, emphasizing the importance of precise initial conditions, discretization techniques, and optimization of bone-based controls to achieve realistic and efficient simulations. Our results demonstrate the effectiveness of these two systems in generating lifelike and precise hand animations for complex hand-object interactions.

5

Conclusions and Future Work

We present two different systems of synthesizing animations of hand-object interaction. Our first system is designed for catching and throwing objects with a physically based control of a simulated hand, where the nominal controller is straightforward to design with human intuition while the fine control producing successful motion is obtained through reinforcement learning. Our second system is designed for manipulation of thin shells with fingertips in a codimensional soft body simulation, where we use optimization with manually crafted objective function to achieve various sleights of cards.

Even though we did not teach the computer how to do a complete, flawless magic trick, we believe that with these two systems as the foundation, the ultimate goal of making computer mastering hand manipulation in simulation is within reach. These two systems shares the idea of letting the computer to assist the control of simulated human hand

in various hand-object interaction tasks, and thus can be applied in multiple scenarios to make synthesizing hand animation easier and faster.

Besides, although our methods can complete throwing and catching different objects, as well as achieve different in-hand manipulation of thin shells, there are some potential improvements we can seek in the future.

First, the target poses for opening and closing the hand in catching and throwing are set manually. To adapt different objects and tasks, we need to train different policies with manually tuned poses. We consider these poses are learnable by the reinforcement learning agents. Learning in full space, i.e., all possible target poses of fingers that can be simulated without interpenetration by adjusting all finger joints within their joint limits, can be extremely hard due to the curse of dimensionality. A potential way to overcome it is to do a principal component analysis on a set of representative poses, and use a reduced set of coordinates.

Second, our current RL agent only learns how to throw. For the catching part, it is controlled by the manually tuned nominal control. For more complicated tasks, we need to learn a catching policy as well to increase its flexibility. We will need to modify our existing DDPG implementation, by tackling the increased dimensionality in combining catching and throwing actions, to train a throwing policy and a catching policy simultaneously.

Third, for the discretization of thin shell like cards, if all the diagonal edges generated during the triangulation with a regular grid are parallel, the card can be bend easier in the direction perpendicular to the parallel diagonal edges and harder in the direction along the diagonal edges. Making the diagonal edges taking different directions can possibly resolve this problem, but requires further experiments.

Forth, for the optimization of card manipulation, the objective function is hard to design manually. The primitives that should be sampled to calculate the objective function have to be chosen carefully, and the objective function has to be meaningful for each in-

dividual sleight. A possible solution might be letting the artificial intelligent to perceive the magic trick and then design the objective functions. Hence, there is much more to be explored in this direction.

Overall, the results validate the effectiveness of the our systems for simulating complex hand tasks. The RL-based system excels in dynamic and repetitive tasks, while the optimization-based controller is well-suited for precise, detailed manipulations. Future work could explore integrating these approaches, leveraging the strengths of both to create even more versatile and robust hand animation systems.

In addition, there are other interesting tasks we would like to try in the future too. For example, interactions between multiple hands, juggling different objects, and ultimately teaching the computer to do a flawless card magic trick.

Bibliography

Sylvain Abondance, Clark B. Teeple, and Robert J. Wood. 2020. A Dexterous Soft Robotic Hand for Delicate In-Hand Manipulation. *IEEE Robotics and Automation Letters* 5, 4 (2020), 5502–5509. https://doi.org/10.1109/LRA.2020.3007411

Sheldon Andrews and Paul G. Kry. 2013. Goal Directed Multi-finger Manipulation: Control Policies and Analysis. *Computers and Graphics* 37, 7 (2013), 830–839. https://doi.org/10.1016/j.cag.2013.04.007

Boris Belousov, Gerhard Neumann, Constantin A. Rothkopf, and Jan Peters. 2016. Catching heuristics are optimal control policies. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (Barcelona, Spain) *(NIPS'16)*. Curran Associates Inc., Red Hook, NY, USA, 1434–1442. https://dl.acm.org/doi/10.5555/3157096.3157257

Jan Bender, Kenny Erleben, and Jeff Trinkle. 2014. Interactive Simulation of Rigid Body Dynamics in Computer Graphics. *Computer Graphics Forum* 33, 1 (2014), 246–270. https://doi.org/10.1111/cgf.12272 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12272

Samuel R. Buss and Jin-Su Kim. 2005. Selectively Damped Least Squares for Inverse Kinematics. *Journal of Graphics Tools* 10, 3 (2005), 37–49. https://doi.org/10.1080/2151237X.2005.10129202

Jason Chemin and Jehee Lee. 2018. A Physics-Based Juggling Simulation Using Reinforcement Learning. In *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games* (Limassol, Cyprus) *(MIG '18)*. Association for Computing Machinery, New York, NY, USA, Article 3, 7 pages. https://doi.org/10.1145/3274247.3274516

Tao Chen, Jie Xu, and Pulkit Agrawal. 2022. A System for General In-Hand Object Re-Orientation. In *Proceedings of the 5th Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 164)*, Aleksandra Faust, David Hsu, and Gerhard Neumann (Eds.). PMLR, London, UK, 297–307. https://proceedings.mlr.press/v164/chen22a.html

Kai Ding, Libin Liu, Michiel van de Panne, and KangKang Yin. 2015. Learning Reduced-Order Feedback Policies for Motion Skills. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (Los Angeles, California) *(SCA '15)*. Association for Computing Machinery, New York, NY, USA, 83–92. https://doi.org/10.1145/2786784.2786802

George ElKoura and Karan Singh. 2003. Handrix: Animating the Human Hand. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Diego, California) *(SCA '03)*. Eurographics Association, Goslar, DEU, 110–119. https://dl.acm.org/doi/10.5555/846276.846291

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative Adversarial Networks. *Commun. ACM* 63, 11 (oct 2020), 139–144. https://doi.org/10.1145/3422622

Nikolaus Hansen. 2006. *The CMA Evolution Strategy: A Comparing Review*. Springer Berlin Heidelberg, Berlin, Heidelberg, 75–102. https://doi.org/10.1007/3-540-32494-1_4

Jean Hugard and Frederick Braue. 1974. *Expert Card Technique*. Dover Publications, New York, NY, USA. https://books.google.ca/books?id=oKpzDQAAQBAJ

Sumit Jain and C. Karen Liu. 2009. Interactive Synthesis of Human-Object Interaction. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New Orleans, Louisiana) *(SCA '09)*. Association for Computing Machinery, New York, NY, USA, 47–53. https://doi.org/10.1145/1599470.1599476

Sumit Jain and C. Karen Liu. 2011a. Controlling Physics-Based Characters Using Soft Contacts. In *Proceedings of the 2011 SIGGRAPH Asia Conference* (Hong Kong, China) *(SA '11)*. Association for Computing Machinery, New York, NY, USA, Article 163, 10 pages. https://doi.org/10.1145/2024156.2024197

Sumit Jain and C. Karen Liu. 2011b. Controlling Physics-Based Characters Using Soft Contacts. *ACM Trans. Graph.* 30, 6 (Dec. 2011), 1–10. https://doi.org/10.1145/2070781.2024197

Jordan Juravsky, Yunrong Guo, Sanja Fidler, and Xue Bin Peng. 2022. PADL: Language-Directed Physics-Based Character Control. In *SIGGRAPH Asia 2022 Conference Papers* (Daegu, Republic of Korea) *(SA '22)*. Association for Computing Machinery, New York, NY, USA, Article 19, 9 pages. https://doi.org/10.1145/3550469.3555391

Shinya Kajikawa, M. Saito, Kohtaro Ohba, and Hikaru Inooka. 1999. Analysis of human arm movement for catching a moving object. In *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028)*, Vol. 2. IEEE, Tokyo, Japan, 698–703 vol.2. https://doi.org/10.1109/ICSMC.1999.825346

Nam Hee Kim, Hung Yu Ling, Zhaoming Xie, and Michiel van de Panne. 2021. Flexible Motion Optimization with Modulated Assistive Forces. *Proc. ACM Comput. Graph. Interact. Tech.* 4, 3, Article 35 (sep 2021), 25 pages. https://doi.org/10.1145/3480144

Seungsu Kim, Ashwini Shukla, and Aude Billard. 2014. Catching Objects in Flight. *IEEE Transactions on Robotics* 30, 5 (2014), 1049–1065. https://doi.org/10.1109/TRO.2014.2316022

Jens Kober, Matthew Glisson, and Michael Mistry. 2012a. Playing catch and juggling with a humanoid robot. In *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. IEEE, Osaka, Japan, 875–881. https://doi.org/10.1109/HUMANOIDS.2012.6651623

Jens Kober, Katharina Muelling, and Jan Peters. 2012b. Learning throwing and catching skills. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Vilamoura-Algarve, Portugal, 5167–5168. https://doi.org/10.1109/IROS.2012.6386267

Paul G. Kry and Dinesh K. Pai. 2006. Interaction capture and synthesis. *ACM Trans. Graph.* 25, 3 (2006), 872–880. https://doi.org/10.1145/1141911.1141969

Ariel Kwiatkowski, Eduardo Alvarado, Vicky Kalogeiton, C. Karen Liu, Julien Pettré, Michiel van de Panne, and Marie-Paule Cani. 2022. A Survey on Reinforcement Learning Methods in Character Animation. *Computer Graphics Forum* 41, 2 (2022), 613–639. https://doi.org/10.1111/cgf.14504

Roberto Lampariello, Duy Nguyen-Tuong, Claudio Castellini, Gerd Hirzinger, and Jan Peters. 2011. Trajectory planning for optimal robot catching in real-time. In *2011 IEEE International Conference on Robotics and Automation*. IEEE, Shanghai, China, 3719–3726. https://doi.org/10.1109/ICRA.2011.5980114

Lei Lan, Guanqun Ma, Yin Yang, Changxi Zheng, Minchen Li, and Chenfanfu Jiang. 2022. Penetration-free projective dynamics on the GPU. *ACM Trans. Graph.* 41, 4, Article 69 (jul 2022), 16 pages. https://doi.org/10.1145/3528223.3530069

Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. 2019. Scalable muscle-actuated human simulation and control. *ACM Trans. Graph.* 38, 4, Article 73 (jul 2019), 13 pages. https://doi.org/10.1145/3306346.3322972

Mengcheng Li, Liang An, Hongwen Zhang, Lianpeng Wu, Feng Chen, Tao Yu, and Yebin Liu. 2022b. Interacting Attention Graph for Single Image Two-Hand Reconstruction. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 2751–2760. https://doi.org/10.1109/CVPR52688.2022.00278

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020. Incremental potential contact: intersection-and inversion-free, large-deformation dynamics. *ACM Trans. Graph.* 39, 4, Article 49 (aug 2020), 20 pages. https://doi.org/10.1145/3386569.3392425

Minchen Li, Danny M. Kaufman, and Chenfanfu Jiang. 2021. Codimensional incremental potential contact. *ACM Trans. Graph.* 40, 4, Article 170 (jul 2021), 24 pages. https://doi.org/10.1145/3450626.3459767

Peizhuo Li, Kfir Aberman, Zihan Zhang, Rana Hanocka, and Olga Sorkine-Hornung. 2022a. GANimator: neural motion synthesis from a single sequence. *ACM Trans. Graph.* 41, 4, Article 138 (jul 2022), 12 pages. https://doi.org/10.1145/3528223.3530157

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2019. Continuous control with deep reinforcement learning. arXiv:1509.02971 [cs.LG] https://arxiv.org/abs/1509.02971

C. Karen Liu. 2009. Dextrous Manipulation from a Grasping Pose. *ACM Trans. Graph.* 28, 3, Article 59 (July 2009), 6 pages. https://doi.org/10.1145/1531326.1531365

Libin Liu and Jessica Hodgins. 2018. Learning Basketball Dribbling Skills Using Trajectory Optimization and Deep Reinforcement Learning. *ACM Trans. Graph.* 37, 4, Article 142 (July 2018), 14 pages. https://doi.org/10.1145/3197517.3201315

Yunhao Luo, Kaixiang Xie, Sheldon Andrews, and Paul Kry. 2021. Catching and Throwing Control of a Physically Simulated Hand. In *Proceedings of the 14th ACM SIGGRAPH Conference on Motion, Interaction and Games* (Virtual Event, Switzerland) *(MIG '21)*. Association for Computing Machinery, New York, NY, USA, Article 15, 7 pages. https://doi.org/10.1145/3487983.3488300

Igor Mordatch, Zoran Popović, and Emanuel Todorov. 2012a. Contact-invariant optimization for hand manipulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Lausanne, Switzerland) *(SCA '12)*. Eurographics Association, Goslar, DEU, 137–144. https://dl.acm.org/doi/10.1145/2185520.2185539

Igor Mordatch, Emanuel Todorov, and Zoran Popović. 2012b. Discovery of complex behaviors through contact-invariant optimization. *ACM Trans. Graph.* 31, 4, Article 43 (jul 2012), 8 pages. https://doi.org/10.1145/2185520.2185539

NASA. 1995. *Man-Systems Integration Standards*. National Aeronautics and Space Administration. https://msis.jsc.nasa.gov/sections/section03.htm

Yoshihiko Ozaki, Yuki Tanigaki, Shuhei Watanabe, and Masaki Onishi. 2020. Multiobjective tree-structured parzen estimator for computationally expensive optimization problems. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (Cancún, Mexico) *(GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 533–541. https://doi.org/10.1145/3377930.3389817

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills. *ACM Trans. Graph.* 37, 4, Article 143 (jul 2018), 14 pages. https://doi.org/10.1145/3197517.3201311

Nancy S. Pollard and Victor Brian Zordan. 2005. Physically Based Grasping Control from Example. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Los Angeles, California) *(SCA '05)*. Association for Computing Machinery, New York, NY, USA, 311–318. https://doi.org/10.1145/1073368.1073413

Seyed Sina Mirrazavi Salehian, Mahdi Khoramshahi, and Aude Billard. 2016. A Dynamical System Approach for Softly Catching a Flying Object: Theory and Experiment. *IEEE Transactions on Robotics* 32, 2 (2016), 462–471. https://doi.org/10.1109/TRO.2016.2536749

Breannan Smith, Chenglei Wu, He Wen, Patrick Peluse, Yaser Sheikh, Jessica K. Hodgins, and Takaaki Shiratori. 2020. Constraining Dense Hand Surface Tracking with Elasticity. *ACM Trans. Graph.* 39, 6, Article 219 (nov 2020), 14 pages. https://doi.org/10.1145/3414685.3417768

Sebastian Starke, Ian Mason, and Taku Komura. 2022. DeepPhase: Periodic Autoencoders for Learning Motion Phase Manifolds. *ACM Trans. Graph.* 41, 4, Article 136 (jul 2022), 13 pages. https://doi.org/10.1145/3528223.3530178

Nkenge Wheatland, Yingying Wang, Huaguang Song, Michael Neff, Victor Zordan, and Sophie Jörg. 2015. State of the Art in Hand and Finger Modeling and Animation. *Computer Graphics Forum* 34, 2 (2015), 735–760. https://doi.org/10.1111/cgf.12595

Andrew Witkin and Michael Kass. 1988. Spacetime constraints. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '88)*. Association for Computing Machinery, New York, NY, USA, 159–168. https://doi.org/10.1145/54852.378507

Cai-Hua Xiong, Wen-Rui Chen, Bai-Yang Sun, Ming-Jin Liu, Shi-Gang Yue, and Wen-Bin Chen. 2016. Design and Implementation of an Anthropomorphic Hand for Replicating Human Grasping Functions. *IEEE Transactions on Robotics* 32, 3 (2016), 652–671. https://doi.org/10.1109/TRO.2016.2558193

Pei Xu and Ioannis Karamouzas. 2021. A GAN-Like Approach for Physics-Based Imitation Learning and Interactive Character Control. *Proc. ACM Comput. Graph. Interact. Tech.* 4, 3, Article 44 (sep 2021), 22 pages. https://doi.org/10.1145/3480148

Yuting Ye and C. Karen Liu. 2012. Synthesis of detailed hand manipulations using contact sampling. *ACM Trans. Graph.* 31, 4, Article 41 (jul 2012), 10 pages. https://doi.org/10.1145/2185520.2185537

Sang Hoon Yeo, Martin Lesmana, Debanga R. Neog, and Dinesh K. Pai. 2012. Eyecatch: Simulating Visuomotor Coordination for Object Interception. *ACM Trans. Graph.* 31, 4, Article 42 (July 2012), 10 pages. https://doi.org/10.1145/2185520.2185538

Baohua Zhang, Yuanxin Xie, Jun Zhou, Kai Wang, and Zhen Zhang. 2020. State-of-the-art robotic grippers, grasping and control strategies, as well as their applications in agricultural robots: A review. *Computers and Electronics in Agriculture* 177 (2020), 105694. https://doi.org/10.1016/j.compag.2020.105694

He Zhang, Yuting Ye, Takaaki Shiratori, and Taku Komura. 2021. ManipNet: Neural Manipulation Synthesis with a Hand-Object Spatial Representation. *ACM Trans. Graph.* 40, 4, Article 121 (jul 2021), 14 pages. https://doi.org/10.1145/3450626.3459830

Appendix

Abbreviations and Glossary of Terms

CFM: constraint force mixing.

Codim-IPC: codimensional incremental potential contact model by Li et al. (2021).

DDPG: deep deterministic policy gradient.

Episode Q0: For agents with a critic, this is the estimate of the discounted long-term reward at the start of each episode, given the initial observation of the environment.

ERP: error reduction parameter.

FEM: finite element method.

FSM: finite state machine.

IK: inverse kinematics

MOTPE: Multi-Objective Tree-structured Parzen Estimator by Ozaki et al. (2020).

Nominal controller: refers to our minimal functional finite state machine controller, which should be assisted by reinforcement learning to achieve its full potential.

PD control: proportional-derivative control

Pareto front (or Pareto frontier): the set of all Pareto efficient trials (i.e. best trials) in a multi-objective optimization.

RL: reinforcement learning.

SLERP: spherical linear interpolation.

Nomenclature of Card Tricks

This nomenclature is based on the classic card magic book by Hugard and Braue (1974).

Card Trick: An individual unit performance in a card magic show. A good card trick has a definite plot, reveal some am using incidents, and has a definite climax.

Sleight of Hand (Card Sleight, or simply Sleight): A series of deceiving hand movements that changes the configuration of the cards.

Card Snapping: Snapping cards apart evenly.

Double Lift: Lifting 2 cards as one.