Catching and Throwing Control of a Physically Simulated Hand

Yunhao Luo
Kaixiang Xie
yunhao.luo@mail.mcgill.ca
kaixiang.xie@mail.mcgill.ca
McGill University
Montreal, QC, Canada

Sheldon Andrews
sheldon.andrews@etsmtl.ca
École de technologie supérieure
Montreal, QC, Canada

Paul Kry
kry@cs.mcgill.ca
McGill University
Montreal, QC, Canada

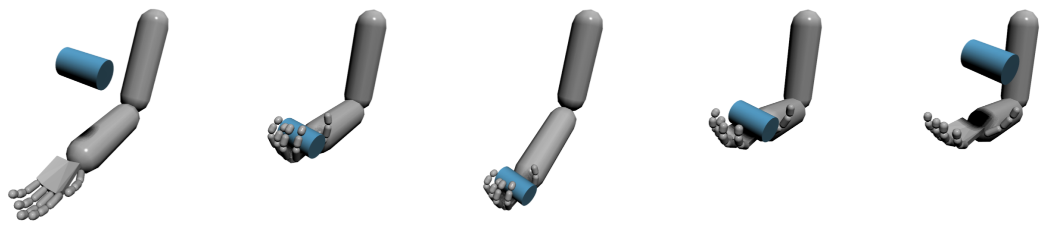

Figure 1: Our physics based controllers start with an intuitive specification of the base motion, and use reinforcement learning to fine tune parameters for successful control of motions such as the throwing and catching of a can seen here.

ABSTRACT

We design a nominal controller for animating an articulated physics-based human arm model, including the hands and fingers, to catch and throw objects. 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. We also discuss ways to extend our approach so that more challenging tasks, such as juggling, may be accomplished.

CCS CONCEPTS

• **Computing methodologies** → **Physical simulation**; *Machine learning*.

KEYWORDS

hand simulation, physics-based animation, grasping, throwing, catching

ACM Reference Format:
Yunhao Luo, Kaixiang Xie, Sheldon Andrews, and Paul Kry. 2021. Catching and Throwing Control of a Physically Simulated Hand. In *Motion, Interaction and Games (MIG '21), November 10–12, 2021, Virtual Event, Switzerland*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3487983.3488300

1 INTRODUCTION

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.

2 RELATED WORK

Synthesis of throwing and catching 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. In comparison, we stress the involvement of the fingers, because we believe the posture of fingers and the orientation of the palm are critical to successful catching and throwing. 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, while not specifically focusing on catching and throwing, has demonstrated that low-dimensional feedback controllers are often sufficient for performing simple ball hitting tasks [Ding et al. 2015]. Throwing and catching is also related to the task of dribbling basketballs, which requires precise and agile control. Liu and Hodgins [2018] 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 RL 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.

Throwing and juggling skills 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.

In contrast to the work discussed here, we do not target robots or build on known neuromuscular control models. Instead, fine adjustments necessary for successful control of an action are learned starting from an easily specified nominal plan for the control.

3 MODEL AND METHODS

Our hand model includes a fully articulated arm and hand with 32 degrees of freedom (DOF) in total, as shown in Figure 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 50th 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). 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.

3.1 Controller Structure

The control scheme is defined as a finite state machine as shown in Figure 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.

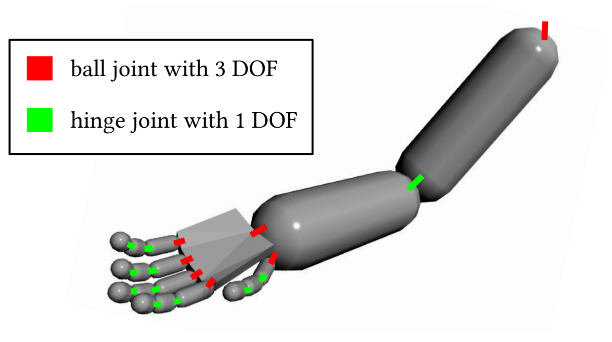

Figure 2: Arm and hand with fingers in its neutral pose.

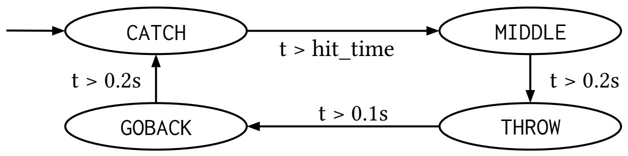

Figure 3: The finite state machine used by our controllers.

Whereas the hand posture is tracked using PD control and a set of predefined poses. 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.
- 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 3.7.
- GOBACK: The hand goes back to a position which is suitable for the next catch. See Section 3.6.

The timing of the transition between states is defined as in Figure 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 3.3) 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. Conversely, during the CATCH and MIDDLE states, the hand closes when contact between the hand and the projectile is detected.

3.2 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

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

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 3.6. For the THROW state, the parameters are controlled by the reinforcement learning agent and further details are discussed in Section 3.7.

3.3 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 2004] 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

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

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.

3.4 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

$$\tau_i = k_p \underbrace{\left(\tilde{\theta}_i - \theta_i\right)}_{\Delta\theta_i} + k_d \dot{\theta}_i, \tag{3}$$

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

$$k_p = w_p \, {m_{\text{joint}}}^2 + k_{\text{base}} \,, \qquad k_d = w_d \, m_{\text{joint}} \,, \tag{4}$$

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 closed hand PD target to fit a smaller shape than the projectile. Additionally, we expect the learning process should compensate the inaccuracy of PD.

3.5 Hand Pose

For performance reasons, spline interpolation and IK control are not used to control the hand pose due to the large number of DOFs involved. Rather, 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.

3.6 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 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.

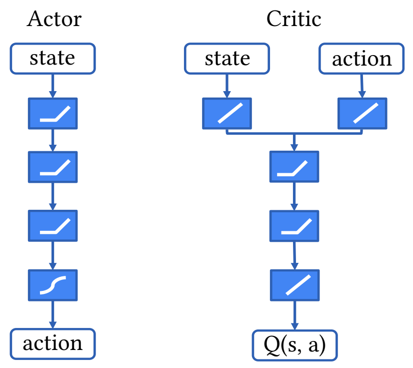

Figure 4: The actor-critic network. Each intermediate layer has 64 nodes. We use tanh **activation for the output of the actor network, and linear activation for the output of the critic network. For the other layers, we use ReLU activation.**

3.7 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 reward function varies from different task (see Section 4).

We use DDPG to train our agent. The structure of the actor and critic networks is shown in Figure 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.

4 RESULTS

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 task, the nominal controller with hand-tuned parameters usually fails at the 2-5th loop of catch-and-throw, but the RL controller with learned parameters appear to be able to loop infinitely.

4.1 Simulation Configuration

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

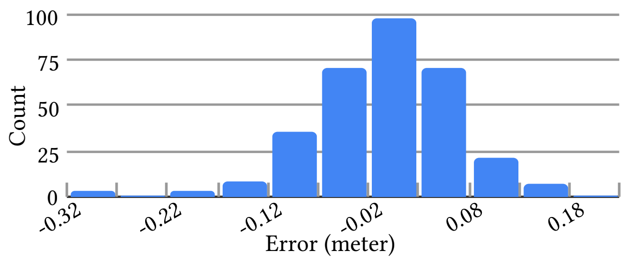

Figure 5: Error distribution of catching and throwing a sphere to the desired height

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

$$ERP = \frac{hk_p}{hk_p + k_d}, \qquad CFM = \frac{1}{hk_p + k_d}.$$

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.

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.

4.2 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 reward function we use is

$$r = \begin{cases} e^{10|y - y_{\text{target}}|} & \text{if the hand catches the sphere} \\ -0.1\,|p_p - p_d| & \text{if the hand misses the sphere} \end{cases} \tag{5}$$

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 $y - y_{target}$ because we want to learn how to deal with different heights.

Before learning, the nominal controller has to be hand-tuned for every different height. With the help of reinforcement learning,

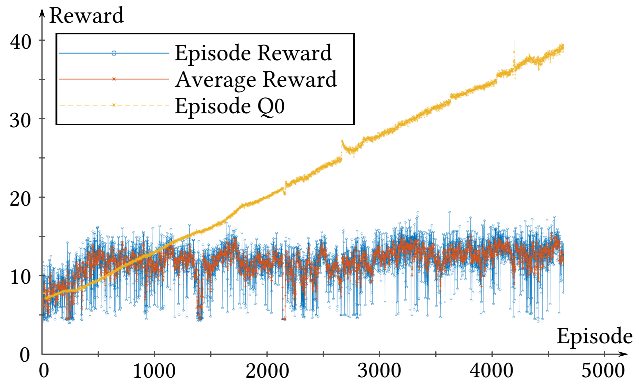

Figure 6: Learning curve of throwing to a desired height.

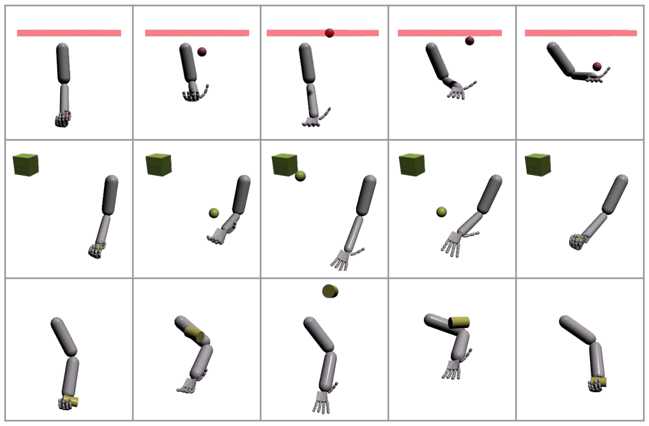

Figure 7: Example motions produced by our controllers. Top, throwing a sphere to the target height denoted by the red plane. Middle, throwing a sphere to hit the target object, and then catching the bounce. Bottom, flipping a can.

the controller can catch and throw the sphere to different heights with learned parameters in real-time. Figure 5 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 within 0.1 m of error. Figure 6 shows the learning curve, where the episode and average reward fluctuates along 10 and the episode Q0 increases linearly. Top row of Figure 7 shows the visual result where the target height is indicated by a transparent red plane.

4.3 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 × 0.1 m × 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

$$r = \begin{cases} 1 & \text{if "hit then catch" is successful} \\ -0.1\,d & \text{otherwise} \end{cases} \tag{6}$$

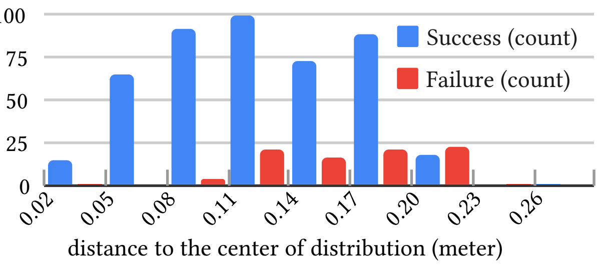

Figure 8: Distance to the center for the position of the box where the agent succeed in or fail the hitting box task.

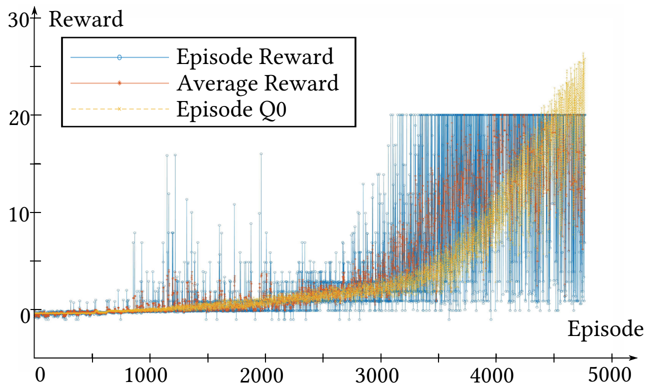

Figure 9: Learning curve of catching and throwing the sphere to hit a target box.

where d is the minimum distance between the sphere and the cube.

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 9 shows the learning curve, where the learning speeds up around the 3000th episode but the rewards greatly fluctuate. Middle row of Figure 7 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 8 shows the distribution of successes and failures in 538 tests. Generally, as the box goes further from the center, the rate of success decreases.

4.4 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

$$r = \begin{cases} e^{10|p_p - p_d|} + r_{\text{flip}} & \text{if "flip then catch" is successful} \\ -0.1\,|p_p - p_d| & \text{otherwise} \end{cases} \quad (7)$$

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 evolution strategy

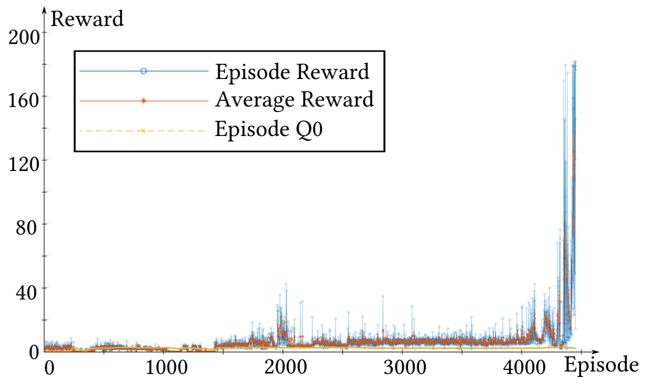

Figure 10: Learning curve of flipping the can.

[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 2nd to 5th 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 7, while the learning curve can be seen in Figure 10.

5 CONCLUSIONS AND FUTURE WORK

We present a system 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.

Although our method can complete throwing and catching different objects, and achieve different goals, there are some potential improvements we can seek in the future.

First, the target poses for opening and closing the hand 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 RL agents. Learning in full space 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 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 the DDPG algorithm to train a throwing policy and a catching policy simultaneously.

There are other tasks we would like to try in the future too. For example, interactions between multiple hands, manipulating multiple objects at the same time, and eventually, juggling different objects.

ACKNOWLEDGMENTS

This project was funded in part by NSERC Discovery 2018-05665.

REFERENCES

S. Andrews and P. 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

B. Belousov, G. Neumann, C. A. Rothkopf, and J. R. Peters. 2016. Catching heuristics are optimal control policies. In *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc., Barcelona, Spain. https://proceedings.neurips.cc/paper/2016/file/43fa7f58b7eac7ac872209342e62e8f1-Paper.pdf

S. R. Buss. 2004. Introduction to inverse kinematics with Jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Transactions in Robotics and Automation* 17 (05 2004).

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

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

N. Hansen. 2006. The CMA evolution strategy: a comparing review. In *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, J.A. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 75–102.

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

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

S. Kajikawa, M. Saito, K. Ohba, and H. 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

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

J. Kober, M. Glisson, and M. 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

J. Kober, K. Muelling, and J. 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

R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, and J. 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

C. K. 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

L. Liu and J. 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

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

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

S. S. M. Salehian, M. Khoramshahi, and A. 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

N. Wheatland, Y. Wang, H. Song, M. Neff, V. Zordan, and S. Jörg. 2015. State of the Art in Hand and Finger Modeling and Animation. *Comput. Graph. Forum* 34, 2 (May 2015), 735–760. https://doi.org/10.1111/cgf.12595

Y. Ye and C. K. Liu. 2012. Synthesis of Detailed Hand Manipulations Using Contact Sampling. *ACM Trans. Graph.* 31, 4, Article 41 (July 2012), 10 pages. https://doi.org/10.1145/2185520.2185537

S. H. Yeo, M. Lesmana, D. R. Neog, and D. 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