Learning Control Policies for Virtual Grasping Applications

Sheldon Andrews Paul Kry Doina Precup
School of Computer Science, McGill University

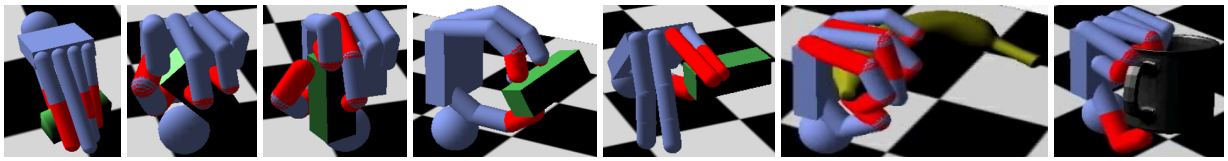

Figure 1: Grasps synthesized by our reinforcement learning framework, including two at right on objects not included in the training episodes.

1 Introduction

Human grasping is one of the most challenging problems in computer animation. Physically based grasp synthesis involves coordination and contact, and depends on many variables such as the shape, size, texture, and physical properties of the target object. The traditional approaches have originated in the robotics community and employ a combination of motion planning (to move within reach of a target) and contact planning (to achieve a stable grasp configuration). For example, the GraspIt! platform [4] has been used to develop such solutions to the grasping problem.

Grasping in computer animation has typically focused on control algorithms that involve the solution of carefully designed optimization problems [2], or in other cases only addresses the reuse of examples under specific initial conditions [5]. In contrast, we present a novel application of reinforcement learning (RL) to grasp synthesis in a physically based virtual environment. A set of basis controllers is used to move the hand along coordinated joint trajectories and a control policy is learned for choosing among them with the goal of automatically synthesizing grasps. Our approach is straightforward and preliminary results show success in learning stable grasps that generalize across objects (see Figure 1).

2 Experimental Setup

We use a hand model (depicted in Figure 2) that approximates the shape and kinematics of a human hand. Each finger has 3 phalanges and 3 joints, corresponding to finger segments and "knuckles", respectively. For simplicity, the visual and physical representations use capsules to model the finger segments and a box for the palm. The joint connecting the first segment of each finger to the palm is modelled as a universal joint, whereas the others are rotary joints with a single degree of freedom.

The state representation is relative to a coordinate system centred at the palm and aligned with the hand (the coordinate system denoted O_H in Figure 2). A palm centric frame is used because the objective is stable grasping and we assume the goal is always to grasp a single object. The state consists of 29 continuous variables:

- $\vec{p}$ - 3D position of the target object in frame O_H
- $\vec{v}$ - linear velocity of the target object in frame O_H
- $\vec{o}$ - orientation of the target object in frame O_H (Euler angles)
- $\theta_{i,j}$ - hinge angle of the jth joint on the ith finger
- ϕ_i - abduction angle of finger i

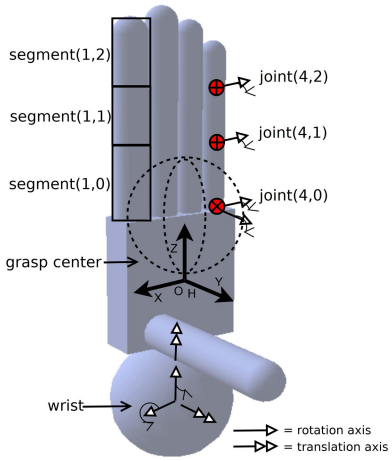

Figure 2: The hand model with reference frame, O_H, centred at the palm. A few selected finger segments are highlighted to show ordering, and joint rotational axes are shown, including the wrist. The wrist axis is shown (lower) and translation directions are denoted using double arrowheads.

The hand and graspable objects are simulated using a physics engine (CMLabs Vortex). Hand posture is managed by a PD controller (with stiffness resembling that of a human hand) that actuates joints according to coordinated motion of joint angles or joint velocities, e.g., open, close, adduct the fingers, pinching pose.

3 Reinforcement Learning

We use a combination of value iteration and SARSA to explore the state-action space and compute an optimal control policy approximation (see [6] for details). The value iteration algorithm begins by selecting one of the user-defined states from an *initiation set*, which is user-defined, and rigorously explores all possible state-action pairs by running the agent until a maximum number of steps n is reached. The initial state set is user-defined and allows the user to direct the agent toward regions of the state space which are relevant. The pseudo-code for the recursive function used to perform value iteration is given in Algorithm 1. Here, $\vec{s}$ represents the current state of the task, as described in Section 2. An action, a, is represented by a PD controller that moves the hand in some coordinated motion, also described in Section 2.

This value iteration step is followed by training episodes of SARSA and ε-greedy exploration, essentially "rounding out" the value function. The capacity to inject strategies into the learned policy is useful for leading the policy toward grasping behaviours

Algorithm 1 valueIteration$(\vec{s}, a, n)$

```
if n > 0 then
    save(s⃗)
    for all a′ ∈ A do
        s⃗′ ← nextState(s⃗, a)
        R ← reward(s⃗, a)
        v ← valueIteration(s⃗′, a′, n − 1)
        Q_{s⃗,a} ← R + γv
    end for
    return max_a Q_{s⃗,a}
end if
return 0
```

that are desired in a given scenario. We provide the user with this ability by making direct changes to the value function, which are further refined in subsequent training episodes leading to desirable behaviours in nearby states.

The value function, $Q_{\vec{s},a}$, is represented using a nearest neighbour function approximator, constructed incrementally during learning (see Algorithm 1). The current state is added to the value function if its distance to the nearest neighbour exceeds a *novel state* threshold η (similar to the trusted state policy used by [1]). Otherwise, the current state is aliased by its nearest neighbour, and the value of its neighbour is updated. At each step, the current state is stored and used to query the agent for an action. The new action is given control of the hand and the simulation is advanced using forward dynamics. Post step, a reward (or penalty) is given to the agent based on the new state.

As recommended by [3], the reward function, $reward(\vec{s}, a)$, is a combination of *immediate* and *delayed* rewards, chosen to accelerate the learning process. Immediate rewards encourage the agent to take actions that will likely lead to good grasping configurations in the short term; delayed rewards, which are given more sparsely, are large bonuses given to the agent when it finally achieves some milestone.

Immediate rewards. An immediate reward, R_p, is given when the agent moves toward the *position* of the target object and is calculated as the decrease in Euclidean distance between the target object and grasping centre compared to the previous time step. The reward R_c is given when the agent chooses an action that increases the number of finger segments in *contact* with the target object; a penalty is given if the number of finger segments is reduced. The value is simply equal to the number of contacts. Typically, large relative velocity between the hand and object will not lead to successful grasping, so penalty, R_v, proportional to the magnitude of the linear *velocity* of the target is given. Another negative reward, R_e, equal to the *effort* required to performing an action, is used to dissuade the agent from choosing irrelevant actions. The effort is estimated as the sum of the magnitudes of the torques at each joint integrated over the simulation time step.

Delayed rewards. A large delayed reward, R_q, is given when the hand finally achieves a grasp, and is based on the *quality* of the grasp. The reward is calculated as the minimum distance from the origin of the set of *wrench* vectors (combined force and torque) due to contact between the hand and the target object. The grasp is stable when the *convex hull* of the available contact wrenches contains the origin of the wrench space; the ability of the grasp to resist perturbations improves as the distance from the origin to the surface of the hull increases. We filter transient grasps by disregarding the grasp quality metric when the velocity of the target object projected onto the palm-aligned axis (see Figure 2) is positive.

The total reward R is calculated at every simulation time step as

$$R = w_p R_p + w_c R_c + w_q R_q - w_v R_v - w_e R_e,$$

where the w factors denote the weight of the reward components. For our experiments, we used $w_p = 0.1$, $w_c = 0.5$, $w_q = 1000$, $w_v = 0.2$, and $w_e = 0.001$. The weights for the reward function were manually adjusted until the agent performed well during preliminary trials. This was largely a trial-and-error process based on observation and required some intuition.

4 Results

Our experiments were performed using a 2.6 GHz Intel quad-core processor and 4 GB of memory. The time required to run a typical episode was ≈ 2 seconds. During the offline learning stage, the simulation is allowed to run faster than real-time, achieving frame rates of $120-500$ frames per second and a mean of ≈ 250 frames per second.

The average time required to query the control policy was ≈ 2.9 ms, and involved performing a nearest neighbour search of about 10000 states. The average time for computing the grasp quality reward R_q was 21.4ms.

Figure 1 shows grasps synthesized using our method, including two experiments (at right) where the policy was tested with objects not seen in the training episodes. The agent was trained with an initiation set of 6 states using the test object (green box). After freezing the control policy, the agent was tested using a series of unseen test objects, in this case, a coffee mug and a banana. The agent was able to successfully grasp each of these objects, however a limitation of the approach is that the agent sometimes produces grasps that are either marginally stable or aesthetically awkward.

5 Conclusion

The preliminary results suggest that it is possible for an RL agent to learn a control policy enabling a virtual hand to do stable grasping of target objects. The agent successfully controlled the hand to perform grasp synthesis, including grasping moving objects and objects of unseen shape and size. While the learning process is automated (the user need only select an initiation set), it also allows for user interaction at different stages through the injection of example strategies.

Generating a robust control policy involves protracted computing time. Our training times could be improved by increasing the efficiency of the convex hull algorithm used to calculate the grasp quality, and the value iteration and SARSA algorithm could be parallelized and offloaded to multiple CPUs. Learning the reward function, e.g., by inverse reinforcement learning, may lead to better trade-offs between grasp quality and other aspects of the performance, and allow for variations on the grasping task. Finally, to improve the aesthetic quality of the synthesized motion, we are developing methods to generate controllers from a motion capture corpus.

Acknowledgements

This work was supported in part by grants from NSERC and GRAND.

References

[1] S. Coros, P. Beaudoin, and M. van de Panne. Robust task-based control policies for physics-based characters. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 28(5), 2009.

[2] C. K. Liu. Dextrous manipulation from a grasping pose. *ACM Transactions on Graphics*, 28(3):1–6, 2009.

[3] M. J. Mataric. Reward functions for accelerated learning. In *Proc. 11th International Conference on Machine Learning*, pages 181–189, 1994.

[4] A. Miller and P. Allen. Graspit! a versatile simulator for robotic grasping. *IEEE Robotics and Automation Magazine*, 11(4):110–122, 2004.

[5] N. S. Pollard and V. B. Zordan. Physically based grasping control from example. In *Proc. 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 311–318, 2005.

[6] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.