Toss Juggling Robots in-sim (and on Hardware)

Sidney Nimako
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA
snimakob@andrew.cmu.edu

***Abstract*—We examine methods for the control and design of a bi-manual dynamic toss juggling robot. We leverage simulation to learn a control policy towards this task as well as a method for learning object dynamics. We observe promising results in both areas. We also present a high-level component selection for hardware to embody the simulated controller.**

***Index Terms*—robotics, manipulation, bimanual, BLDC, MuJoCo, DDPG, VAE, Codesign**

I. Introduction

Juggling is a difficult skill even for humans to perform. It requires an intuition for object dynamics, fine force control, and rapid, accurate movements.

Juggling robots have been constructed in the past, but they often functioned using bounce-juggling a distinct manipulation method that does not generalize well for other tasks. Few robots have demonstrated the ability to catch and toss repeatedly, and the ones that do, often apply bespoke hardware for the juggling task.

Controls and hardware developed for bi-manual juggling could expand to other dynamic tasks such as chopping, drumming, and hammering.

We contribute a design and controller for a bi-manual juggling robot based on ubiquitous brushless DC motors. We employ a co-design framework using simulation results to inform the selection and configuration of actuators and the availability of actuators, and other components, to inform the parameters used in the simulation.

We will explore the following:

- Leveraging a deep deterministic policy gradient (DDPG) network to complete the juggling task
- Developing a variational auto-encoder (VAE) to learn embeddings for object characteristics that affect dynamics
- Employing co-design principles, to specify hardware to embody the system using commercial off-the-shelf components.

II. Methods

We heavily utilized simulation for all of the contributions in this paper. We elected to use Google's MuJoCo framework for simulating the dynamics of the juggling environment. MuJoCo provides a method for both modeling and simulating the dynamics of the ball and juggling arms. Additionally, it is compatible with Gymnasium, a machine learning framework originally developed by OpenAI and currently maintained by the Farama Foundation.

The simulation and training of the various networks were run locally on a Dell XPS 9520 with a 12th Gen Intel(R) Core(TM) i9-12900HK (2.50 GHz) and 16.0 GB of RAM. The system also contained an Nvidia GeForce RTX 3050 Ti Mobile graphics processor. All operations were done in Ubuntu 22.03.

A. Deep-deterministic Policy Gradient

Deep-Deterministic Policy Gradients or DDPGs are a reinforcement learning model used to learn a mapping from a continuous observation space to a continuous action space. The model builds on the actor-critic framework to optimize the policy under observed rewards. For stability, our implementation also incorporated a target network. We implemented our DDPG using Pytorch and connected the simulation discussed above through Gymnasium. We evaluated toss juggling and trained our network in several passes. In the initial configuration we used a single arm and evaluated the ability to learn a policy resulting in successful catches of a ball dropped within the configuration space of the arm. This scenario was also used in the co-design process to determine the configuration of the arms' joints, as well as link lengths, motor torque, and speed.

For the catching environment, the reward function was as follows:

$$r_{ctrl} = -a^T a \tag{1}$$

$$r_d = -||x_{ball} - x_{palm}|| \tag{2}$$

$$r_{catch} = \mathbb{1}(||x_{ball} - x_{palm}|| < 0.15)1000 \tag{3}$$

$$\gamma = 0.1 \tag{4}$$

$$r = r_d + \gamma r_{ctrl} + r_{catch} \tag{5}$$

The environment terminated on catch or if the ball dropped below the catchable height. We successfully learned a policy capable of 500 consecutive catches with randomized initial ball positions.

We proceeded to a bimanual environment with the system learning to catch and then toss the ball. Both arms in this environment were identical to the previous. The following modifications were made for this environment. Firstly, we remove the termination condition when the ball is caught. Then we augment the rewards function with a state-based penalty. Using the same condition as before to indicate a catch, we begin applying a penalty while that state is true.

$$r_{hold} = \mathbb{1}(||x_{ball} - x_{palm}|| < 0.15)10$$

Using this reward space, the policy learned was opportunistic. The arm learned to bump the ball with the outside of the hand rather than catching it. To address this we shifted to a contact-based model for determining catch completion. We also adjusted the rewards function to reward high tosses and added a one-shot reward for bringing the ball to a standstill. This was to prevent another opportunistic policy, which manifested in hitting the ball into the air rather than catching and tossing it. The final reward function was as follows:

$$r_{ctrl} = -a^T a \tag{6}$$
$$r_d = -||x_{ball} - x_{palm}|| \tag{7}$$
$$r_{catch} = \mathbb{1}(||x_{ball} - x_{palm}|| < 0.15)1000 \tag{8}$$
$$\gamma = 0.1 \tag{9}$$
$$r_{throw} = \mathbb{1}(thrown)||x_{throw}|| \tag{10}$$
$$x_{throw} = x_{hand} - x_{ball} \tag{11}$$
$$r_{height} = \mathbb{1}(thrown)x_{ball,z} \tag{12}$$
$$r_{hold} = \mathbb{1}(holding)(-15) \tag{13}$$
$$holding = \text{IN CONTACT AND ALREADY CAUGHT} \tag{14}$$
$$r_{ss} = \mathbb{1}(v_{ball,z} < 0.01)1000 \tag{15}$$
$$r_{movement} = -(||\Delta_{left}|| + ||\Delta_{right}||) \tag{16}$$
$$\tag{17}$$

The rewards similarly optimize for proximity to the ball while it is airborne. Once caught it penalizes for holding and rewards for high tosses and keeping the arm close to the initial position. This further incentivizes a tossing motion as opposed to lifting the ball to a higher position.

Once we had learned a policy capable of consistently catching and tossing the ball (200 consecutive catches and tosses), we began training the model to catch the ball and toss it to the other hand. The rewards function stayed the same, and the environment was wrapped in a state machine to govern which arm was tossing and which was catching. At this point, computational bottlenecks made the problem intractable. The simulation began to take several seconds per iteration and with the large action space, adequate exploration could not be completed promptly.

Nevertheless, we did begin to see behavioral convergence toward the goal after 10000 iterations. The ball was successfully caught and tossed towards the other arm, however, we could not complete a subsequent catch.

B. Learning object embeddings

To learn the object embeddings we collected 1000s of samples of simulated data with various mass configurations of the ball. Using MuJoCo, we collected data on the ball's position and velocity in a ballistic state, using randomized initial positions and velocities. We trained a symmetric 4-layer encoder-decoder network to reduce a set of 260 observations into a representation of the object consisting of 10 values. The network consisted of 4 linear layers with ReLu activation functions except for the output of both the encoder and decoder which was not activated.

Once training was complete we had an average mse loss of 0.0912 over 100 test data points.

Using the single previous observation and this representation we evaluated the utility of the system as a dynamics approximater by predicting the next state. The regression network we used was a 3-layer neural network with ReLu activation functions.

Once training was complete we had a loss of 0.01 in our regression model.

We generated a new set of data samples using an unseen mass configuration. We computed the encoder representation using a single window of the new scenario and evaluated the regression loss.

We achieved an average loss of 0.05. This shows promise as a method for learning object dynamics, however further study is needed to validate its efficacy. We propose examining more complex scenarios and establishing a baseline system for dynamic predictions for comparison.

III. Hardware Co-design

We design the system as a pair of identical arms. Each consists of three identical actuators and carbon fiber tubing serving as linkages.

Our initial, single-arm exploration was used to determine the minimum number of actuators necessary to complete the task. Decreasing the number of actuators reduces the cost of developing the system as the actuators and control electronics are the bulk of the cost. We also explored the space of joint orientations.

We selected a ball with a mass of 100 g for simulation purposes and tuned the resulting motor parameters to allow for adequate manipulation. With identical tubing lengths of 0.2 m, to toss the ball 0.5 meters into the air we would need an exit velocity of about 3 m/s. To achieve this with a radius of 0.4 meters we would need 25 radians per second or approximately 240 rpm. To hold the mass at full arm extension we would also need an output torque of 0.4 Nm. In practice, it is unlikely that the ball will be tossed at full arm extension so target a motor speed of 300 rpm.

Therefore we need a motor and gear system that reaches 0.4 Nm while being below its stall torque. We selected the MN2806 Antigravity motor. At max power consumption, it can output 0.15 Nm with 6235 rpm. Targeting 50% power usage and selecting an appropriate gear ratio we get 375.5 rpm at 0.6 Nm with a 20:1 gear ratio. Not only do these values fall above our targets, but we have an additional 50 percent of the total power consumption available, allowing for possible manipulation of heavier objects at faster speeds. To reduce backlash on impact we select a harmonic gear system for our gearbox. Hobbyists have successfully manufactured harmonic gearboxes using flexible filament and 3D printers.

IV. Conclusion

We apply a deep-deterministic policy gradient network to train a simulated juggling environment. We engineer a rewards function that approximates our desired behavior and train the network with promising results. We successfully learned a policy that can consistently catch and toss a dropped ball. We are confident with further development single ball tossing and catching, is achievable. Additionally, we explore designs for embodying this policy using commercial off-the-shelf components. We use the simulation and part availability to determine simulation parameters and simulation results to inform design choices. Finally, we attempt to learn embeddings of object dynamics using a variational autoencoder. We successfully generated an encoder and predictor system that can accurately predict ball trajectory using a single previous point. Further testing is necessary to determine the generality of this system.